Пояснювальна записка
до дипломного проекту
Кульчицького Олексія Андрійовича
на тему: «Система віддаленого планування оповіщень»

Київ – 2019 рік

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Гб — Гігабайт;

ООП — Об’єктно-орієнтоване програмування;

СУБД — Система управління базами даних;

API — Application program interface;

AWS — Amazon web services;

CQRS — Command and query responsibility segregation;

EF — Entity framework;

HTML — Hyper Text Markup Language;

HTTP — Hyper Text Transfer Protocol;

IP — Internet Protocol;

JSON — JavaScript object notation;

ORM — Object-Relation mapper;

SMTP — Simple mail transfer protocol;

SQL — Structured query language;

TCP — Transmission control protocol;

XML — Extensible markup language;

ВСТУП

Оповіщення — один з основних методів комунікації у сучасних програмних системах. Оповіщення використовують як для прямої взаємодії з користувачами системи, так і для взаємодії між її внутрішніми або зовнішніми компонентами.

Актуальність теми

У процесі розробки програмних систем, які дуже часто повинні надсилати різного типу оповіщення, кожен замовник бажає максимально зекономити свої витрати на обчислювальні потужності, інфраструктурну підтримку системи та подальший її супровід. Натомість зазвичай замовники постійно вносять зміни та корективи у технічні і функціональні вимоги замовленого продукту, на які, як вони вважають, не потрібно багато часу та ресурсів. Саме тому створення рішень, які дозволять розробникам та замовникам зменшити кількість витрачених грошей та часу на розробку, тестування і підтримку програмних продуктів є надзвичайно важливою проблемою, над вирішенням якої постійно працюють у всьому світі.

На жаль, з такими проблемами стикається чимала кількість розробників і досить часто причиною цих проблем стає функціонал системи, який має відкладено за певним графіком надсилати кінцевим клієнтам системи або ж стороннім застосункам оповіщення. На такі вимоги завжди витрачається багато часу на проектування та розробку і невиправдані суми грошей, щоб покрити затрати на обчислювальні потужності, оскільки будь-яка реалізація планувальника задач потребує великої кількості ресурсів центрального процесора та оперативної пам’яті сервера.

Мета і задача проектування

Метою дипломного проекту є створення віддаленої, повністю незалежної, безкоштовної та автономної системи для планування і відправки

оповіщень, на яку можна перекласти навантаження, з власної системи, пов’язане з процесом відправки та планування оповіщень за графіком.

Задачею дипломного проекту є аналіз можливих методів реалізації системи, розробка її архітектури, програмування та тестування.

Галузь використання

Система, що буде розроблена у даному проекті, призначена для використання у галузі розробки програмного забезпечення. Кінцевим користувачем системи є розробники та, розроблене ними, програмне забезпечення, яке буде використовувати систему віддаленого планування оповіщень як зовнішній, незалежний модуль.

Структура роботи

У розділі 1 представлено аналіз сучасних методів вирішення проблеми та огляд існуючих рішень.

У розділі 2 визначено технічні та функціональні вимоги до системи.

У розділі 3 оглянуто технології, що будуть використані при розробці.

У розділі 4 представлено опис процесу розробки системи.

У розділі 5 проведено тестування системи.

Додаток А містить структурну схему системи.

Додаток Б містить блок-схему загального алгоритму роботи системи.

Додаток В містить UML діаграму послідовності роботи принципу CQRS.

Додаток Г містить структурну схему бази даних.

Додаток Д містить блок-схему алгоритму авторизації.

Додаток Е містить вихідний код системи.

1. АНАЛІЗ СУЧАСНИХ МЕТОДІВ ПЛАНУВАННЯ ТА ВІДПРАВКИ ОПОВІЩЕНЬ

Перед початком проектування та розробки системи варто розглянути існуючі методи планування та відправки оповіщень, а також визначити основні типи оповіщень, з якими повинна працювати система.

1.1 Визначення основних типів оповіщень

Оскільки поняття оповіщення або ж його синонім сповіщення означає повідомлення, яке доставляється кінцевому отримувачу [1], то спочатку необхідно провести класифікацію типів отримувачів цього повідомлення, за допомогою якої можна буде визначити типи оповіщень, з якими буде працювати система.

Адресатами повідомлень у сучасних програмних системах є не тільки люди, що працюють з цими системами, а й інші системи або ж їх компоненти, які відправляють один одному оповіщення про події, які стались під час їх роботи. Можна виділити чотири типи адресатів:

- користувач системи, що використовує її в особистих цілях, тобто кінцевий користувач, який приносить власнику системи вигоду;
- користувач системи, для якого вона є його робочим середовищем;
- програмний компонент системи, що працює на основі отриманих оповіщень;

Керуючись отриманою класифікацію, можна визначити найбільш популярні типи оповіщень, які використовуються у сучасній розробці. Кінцевим користувачам системи, зазвичай відправляються оповіщення на електронну пошту, у месенджери, а також оповіщення у реальному часі на веб-інтерфейс. Серед них можна обрати найпопулярніший тип оповіщення,

який використовується в абсолютній більшості сучасних систем, — електронний лист.

Серед оповіщень між компонентами систем абсолютним лідером по полярності є повідомлення надіслане за протоколом HTTP [2], адже серверна частина будь-якого застосунку має один або кілька програмних інтерфейсів, які забезпечують працюють на основі протоколу HTTP.

Отже, основними типами оповіщень будуть HTTP-запит та електронний лист.

1.2 Аналіз методів реалізації планування оповіщень

Перед тим як почати аналіз, необхідно визначитись з поняттям планування в контексті даного проекту. Планування оповіщень — це створення та відправка оповіщень відкладено та за встановленим графіком.

Серед сучасних методів реалізації планування задач, якими користуються розробники, можна виділити два основних:

- локальний — реалізація, підтримка і налагодження планувальника власноруч, подальше розгортання локально у рамках вашої системи;
- віддалений — реалізація, налагодження та супровід роботи планувальника передана зовнішньому підряднику, який відповідальний за його якісну та стабільну роботу;

Також можна виділити характеристики обраних методів, які будуть проаналізовані та створити порівняльну характеристику:

- часові затрати на реалізацію;
- проблеми під час супроводу;
- грошові затрати;

Для локального методу реалізації очевидно, що часові затрати на написання коду планувальника та його налагодження, будуть значно вищими

ніж для віддаленого, адже при віддаленому методі використовується готове рішення, яке вже запрограмоване та налагоджене.

Що стосується проблем під час супроводу системи планування, то тут все теж досить очевидно. При локальному методі реалізації, команда розробників, що запрограмувала системи планування, у подальшому буде нести відповідальність за коректність її роботи, виправлення можливих помилок, а також за постійний моніторинг ресурсів сервера, адже будь-яка реалізація планувальника, буде споживати багато ресурсів центрального процесора, оперативної пам’яті та мережі. У випадку ж використання віддаленого методу, розробники, які використають віддалений планувальних, будуть відповідати лише за код, що відповідає за інтеграцію з планувальником та реалізацію своєї бізнес-логіки на його основі, без постійних проблем з використанням ресурсів та виправленням помилок.

У питанні грошових затрат, віддалений метод, на мою думку, теж однозначний лідер, адже всім відомо, що розробка у наші дні коштує чималих грошей, але далеко не всі розуміють, що супровід системи з локальним модулем планування оповіщень, яка зі збільшенням кількості користувачів буде споживати набагато більше ресурсів сервера або ж цілого кластера значно дорожчий ніж супровід системи, у якій це навантаження буде покладене на зовнішній ресурс.

Ознайомившись з порівняльною характеристикою, вибір віддаленого методу планування оповіщень, значно дешевший, швидший та більш привабливий для розробників та замовників.

1.3 Огляд існуючих рішень

На основі проведеного попереднього аналізу були відібрані популярні сучасні сервіси, що використовуються для віддаленого планування

оповіщень. Огляд буде проводитись за двома основними критеріями: функціональні можливості та ціна, яку бере сервіс за надання своїх послуг.

1.3.1 MailChimp

Один з найпопулярніших сервісів відправки електронних повідомлень у світі, який орієнтований на створення та управління масових розсилок повідомлень із можливостями збору та агрегації аналітичних маркетингових даних, що необхідні користувачам для просування та розвитку їх продуктів цілях. Приклад створення маркетингової розсилки наведено на рисунку 1.1.

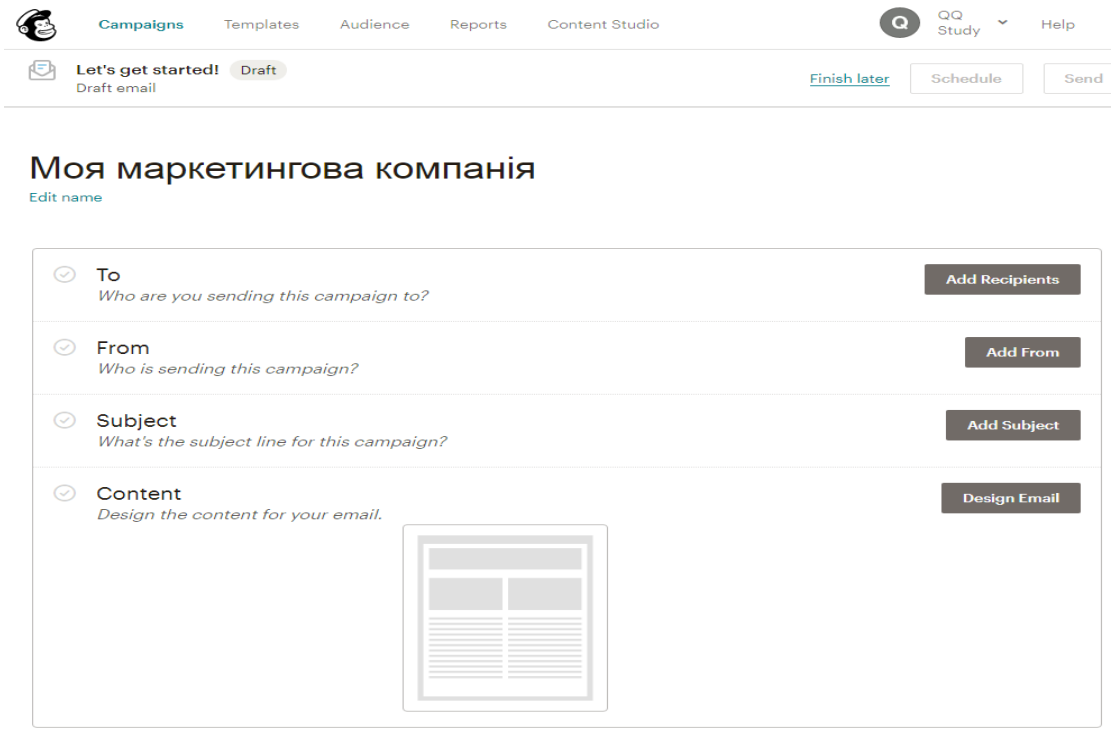

Рисунок 1.1 — Сторінка створення маркетингової кампанії в сервісі «MailChimp»

Для розробників сервіс підготував комплексну онлайн документацію з керівництвом по процесу впровадження і використання.

Основні функціональні можливості:

– можливість об’єднувати оповіщення у групи;
– можливість відслідковувати потрапляння оповіщень до спам фільтрів;
– можливість відкладено відправити оповіщення електронною поштою;
– можливість задати та оновити графік відправки оповіщення;
– можливість використання HTML шаблонізатору контенту електронного листа;
– можливість створення маркетингових кампаній;
– можливість отримання аналітики по групі користувачів, маркетинговій кампанії або в цілому по проекту;
– інтеграція з маркетинговими сервісами «Google Ads» та «Facebook Advertising»;
– авторизація, що базується на стандарті OAuth 2.0 [3];

Ціни розподілені на 4 типи планів:

– «Free» — безкоштовний з обмеженням у 2000 адресатів, ліміт 7 маркетингових кампаній, відсутність можливості отримувати 24/7 підтримку та ліміт у 10000 відправлених повідомлень на місяць;
– «Essentials» — ціна 9.99$ на місяць, включає в себе план «Free», дозволяє користуватись усіма типами шаблонів для електронних листів, має обмеження у 50000 адресатів з лімітом у 500000 повідомлень на місяць та цілодобовою підтримкою;
– «Standard» — ціна 14.99$ на місяці, включає план «Essentials», надає розширені можливості оновлення маркетингових кампаній, має обмеження у 100000 отримувачів та 1200000 відправлених листів на місяць;
– «Premium» — ціна 299.99$ на місяць, надає режим підтримки по телефону, ліміт в 200000 отримувачів та 3000000 відправлених листів на місяць;

Сервіс «MailChimp» дуже добре підійде системам, для яких важливі показники конверсії відправлених оповіщень та їх глибока аналітика, а оскільки він має безкоштовні тарифні плани, то будь-який розробник чи власник бізнесу може швидко та легко спробувати його можливості.

1.3.2 SendGrid

Сервіс планування та відправки електронних листів, який надає можливості для гнучкого планування оповіщень у вигляді електронних листів із детальною статистикою показників доставки, відгуку на повідомлення та аналізом конверсії. Приклад панелі налаштувань наведено на рисунку 1.2.

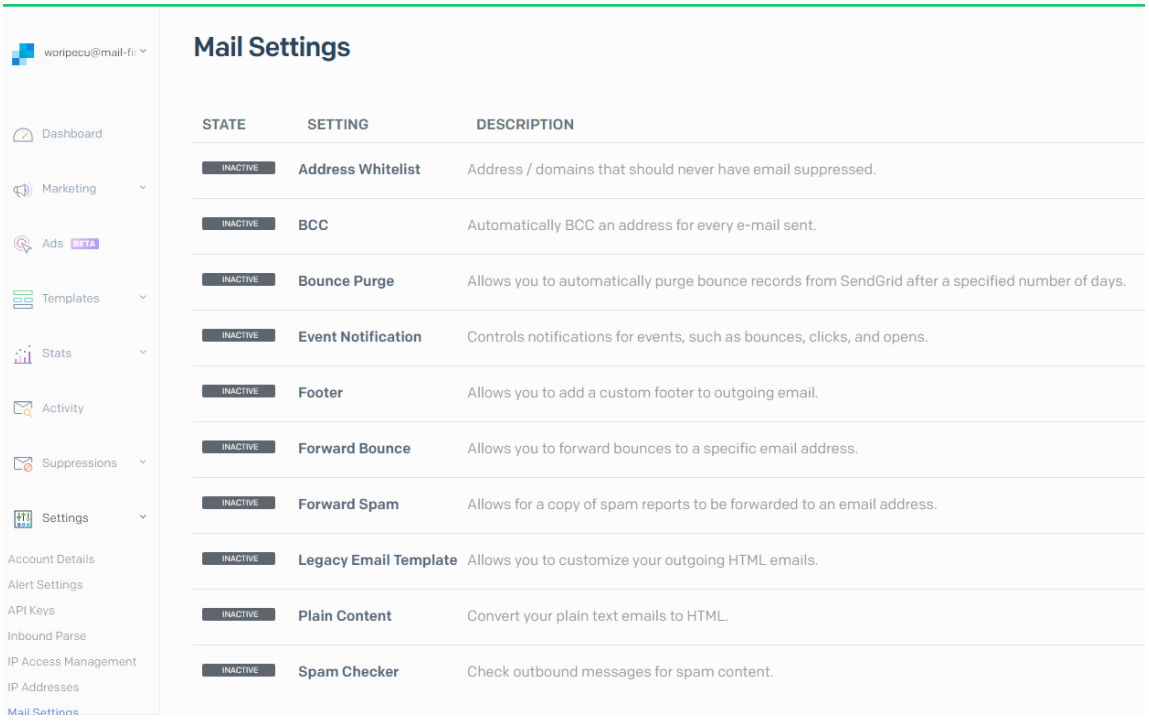

Рисунок 1.2. — Панель налаштування аканту «SendGrid»

Варто зауважити, що користувацького веб-інтерфейсу для відправки повідомлень немає. На сайті сервісу, користувач може змінювати основні налаштування аканту, прав доступу та шаблонів листів.

Для серверної інтеграції сервіс надає не тільки можливість взаємодії за допомогою HTTP протоколу, а також за допомогою протоколу SMTP [4], що дозволяє використовувати його напряму в якості провайдера відправки повідомлень.

Функціональні можливості:

- наявна система авторизації сервер-сервер;
- можливість налаштування списку дозволених для роботи з системою IP-адрес;
- можливість створювати списки заблокованих адрес електронної пошти;
- можливість створювати маркетингові розсилки;
- можливість розділяти оповіщення по категоріям;
- можливість перевірки адреси електронної пошти на існування;
- можливість налаштування оповіщень про відправку оповіщень;
- можливість відкладено надіслати електронний лист;
- можливість оновлення існуючих запланованих оповіщень;
- відсутність інтеграції з маркетинговими сервісами;
- відсутня можливість використовувати кирилицю для назв кампаній та шаблонів;
- за один HTTP запит дозволено відправити не більше тисячі оповіщень для будь-якого тарифного плану;
- незручна система помилок, що повертаються при роботі через серверну інтеграцію;

Сервіс «SendGrid» надає чотири тарифних плани:

- «Free» — безкоштовний. Доступні основні функціональні можливості, але без виділених IP-адрес та створення підлеглих користувачів. Обмеження 40000 повідомлень на перший місяць, далі – 100 повідомлень в день.

– «Essentials» —14.95$ на місяць, дублює можливості «Free» плану, але надає обмеження у 100000 листів щомісяця.
– «Pro» — 79.95$ на місяць, включає план «Essentials», та розширює його лімітом в 1500000 повідомлень на місяць та виділеним SMTP сервером.
– «Premier» — договірна ціна, з лімітом 5 мільйонів повідомлень, цілодобовою підтримкою та системою управління користувачами в рамках одного облікового запису. Включає план «Pro».

«SendGrid», так само як і «MailChimp», підійде системам, в яких важлива маркетингова складова відправлених оповіщень. Також, у порівнянні з «MailChimp», безкоштовний тарифний план має набагато менше обмежень, а також для клієнтів з потребою відправки надзвичайно великої кількості оповіщень надає особливі умови співпраці.

Варто зазначити, що через відсутність можливості безпосереднього використання планування та відправки оповіщень через користувацький інтерфейс, унеможливлює інтеграцію сервісу з системами, які розгорнуті без сервера та керуються людьми.

1.3.3 Mailgun

Сервіс відправки електронних листів, який часто використовують як дешевшу альтернативу «MailChimp» та «SendGrid». Сам сервіс позиціонує себе як систему автоматизації відправки електронних листів, хоча можливості відправляти оповіщення за певним графіком він не надає. Панель керування сервісу «Mailgun» зображена на рисунку 1.3.

З панелі керування можна переглядати статистичні дані по власному обліковому запису, а також налаштовувати особисті дані, такі як: ім’я, прізвище та адреса електронної пошти, та дані необхідні для інтеграції з даним сервісом.

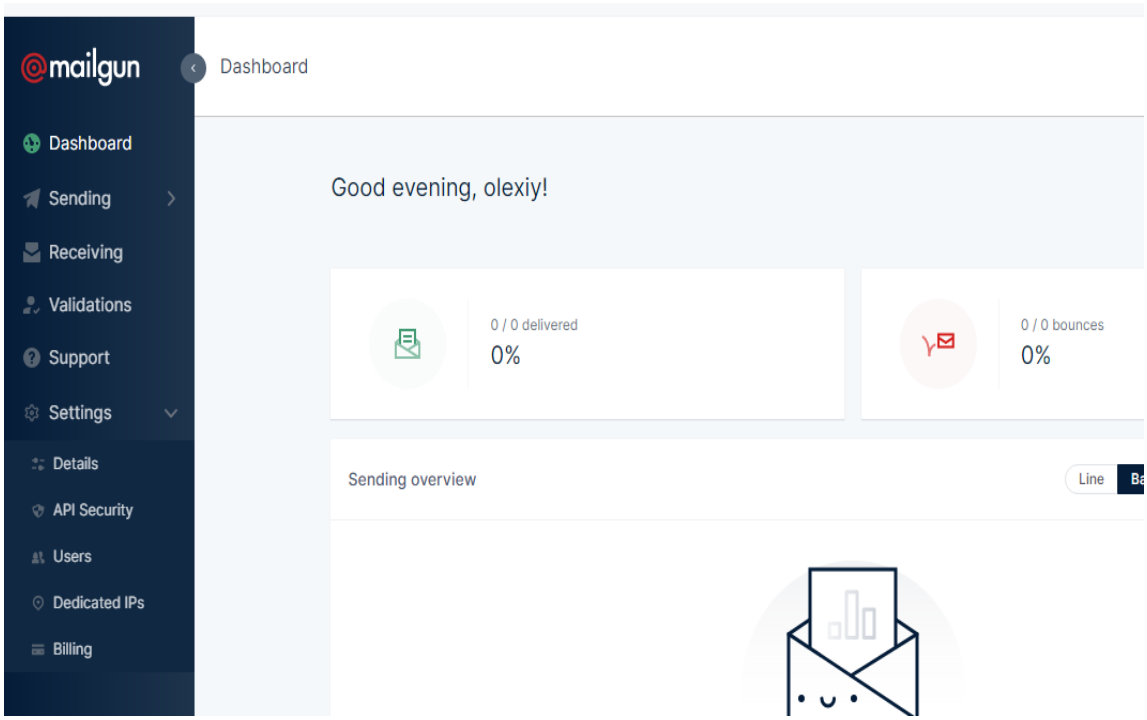

Рисунок 1.3 — Панель керування сервісу «Mailgun»

Функціональні можливості:

– можливість серверної авторизації за допомогою кількох серверних ключів;
– можливість використовувати окремий інтерфейс для валідації адрес електронної пошти, що дуже корисно для фільтрації тимчасових поштових скриньок;
– можливість встановити SMTP агент сервісу на власний сервер;
– немає можливості відкладено відправити оповіщення;
– наявні готові бібліотеки для інтеграції на різних мовах програмування, таких як: C#, Java, Python та Ruby;
– неможливо встановити графік відправки оповіщень;

Плата за сервіс «Mailgun» складається з кількості відправлених повідомлень на місяць та кількості проведених валідацій адрес електронної пошти, при чому перші 10000 відправлених оповіщень та 100 провалідованих адрес на місяць є безкоштовними. Надалі кожні 10000 відправлених повідомлень будуть коштувати 5$. У випадку досягнення

ліміту в 100000 відправлених повідомлень ціна буде зафіксована на відмітці 80$ за додатковою оплатою в 30$ за кожних 50000 повідомлень.

Також варто зазначити гнучкий тарифний план «Enterprise Custom», що дозволяє домовитись із представниками сервісу про особливі умови користування.

Даний сервіс підійде для систем, яким необхідно за короткий термін реалізувати безкоштовну відправку електронних листів, з чим їм допоможуть вже готові бібліотеки для інтеграції.

1.3.4 Amazon Simple Email Service

Сервіс від гіганта індустрії – «Amazon», дещо відрізняється від, описаних вище, аналогів, адже він входить то величезної екосистеми AWS [5], та може бути використаний у парі з абсолютною більшістю сервісів від компанії «Amazon», на рисунку 1.4 зображено консоль керування в контексті облікового запису AWS.

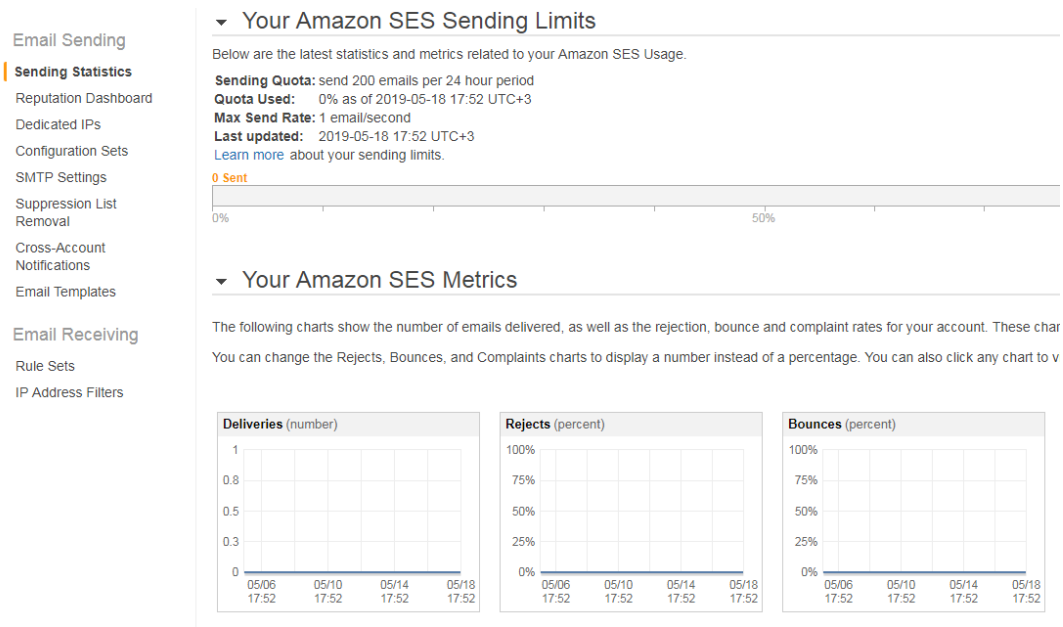

Рисунок 1.4 — Консоль керування «Amazon Simple Email Service»

Функціональні можливості:

- можливість отримання виділеного SMTP сервера;
- можливість налаштовувати права доступу в рамках одного аканту або ж створити групу нових облікових записів;
- можливість динамічно змінювати дата центр для відправки повідомлень;
- відсутня можливість відправити відкладене оповіщення або ж задати графік відправки оповіщень за допомогою HTTP запиту на інтерфейс сервісу, лише через додатковий сервіс «AWS Lamda»[6];

Сервіс «Amazon Simple Email Service» не має чітких тарифів, а користувачі платять по факту використаних ресурсів:

- відправка — безкоштовно перші 62000 оповіщень, надалі 0.1$ за кожну тисячу оповіщень;
- отримання — безкоштовно перші 1000 оповіщень, надалі 0.1$ за кожну тисячу оповіщень;

«Amazon Simple Email Service» найкраще підходить для розробників з досвідом роботи із сервісами «Amazon», які зможуть встановити для нього правильні налаштування безпеки та не втратити у майбутньому на цьому гроші. З точки зору бізнесу даний сервіс привабливий лише через дешеву ціну за його пропускну здатність.

1.3.5 Posthook

Оскільки одним з типів оповіщень у системі планування оповіщень, що розглядається в рамках даного дипломного проекту є HTTP оповіщення, то варто розглянути сервіс, який має схожі функціональні можливості. Особистий кабінет сервісу «Posthook» зображено на рисунку 1.5.

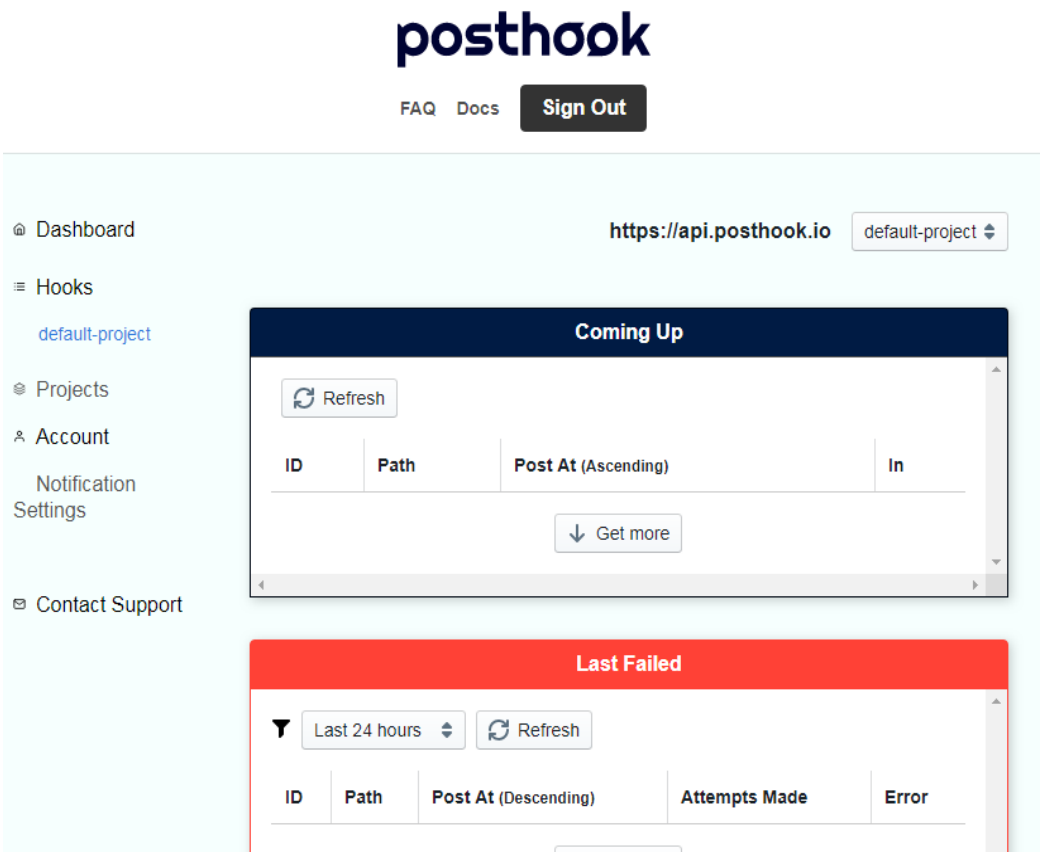

Рисунок 1.5 — особистий кабінет сервісу Posthook

Функціональні можливості:

– можливість відкладено відправити HTTP оповіщення;
– можливість використання тіла HTTP оповіщення в форматі JSON;
– можливість перевірки підпису кожного окремого отриманого оповіщення;
– можливість створити групи для оповіщень;
– можливість переглядати заплановані, виконані та відмінені оповіщення;
– можливість зміни серверного ключа у випадку підозрілої активності;
– можливість увімкнути хеш-підпис для кожного запиту за алгоритмом SHA256 [7];
– можливість налаштування стратегії перевідправки оповіщення, у випадку недоступності сервера-адресата;

Тарифні плани сервісу «Posthook»:

- «Project» — 19$ на місяць, обмеження в 50000 оповіщень на місяць;
- «Startup» — 49$ на місяць, обмеження в 200000 оповіщень на місяць;
- «Corporate» — 129$ на місяць, обмеження 1000000 оповіщень на місяць;

«Posthook» буде дуже корисний для систем, які використовуються, як провайдер послуг, великою кількістю зовнішніх програмних продуктів, яким необхідно постійно відправляти HTTP оповіщення з інформацією про пов’язані з ними зміни в системі.

Висновки до розділу 1

У цьому розділі було проаналізовано типи оповіщень, які використовуються у сучасних програмних системах та обрано два основних: HTTP оповіщення та оповіщення електронною поштою.

Також було розглянуто методи реалізації планування оповіщень сучасними розробниками. Після порівняння локального та віддаленого методів, переваги другого були очевидними.

Для визначених основних типів оповіщень було оглянуто найпопулярніші системи планування та відправки оповіщень, які є аналогами системи, що розглядається у даному дипломному проекті.

Обрати однозначно краще рішення серед них не є можливим, адже кожна розглянута система, хоч і має необхідні базові функціональні можливості, суттєво відрізняється позиціонуванням на ринку: «MailChimp» та «SendGrid» більше орієнтовані на роботу зі оповіщеннями, які потім будуть аналізуватись та на їх основі буде будуватись маркетингова кампанія, у той час як «Posthook», «Mailgun» та «Amazon Simple Email Service» більше орієнтовані на розробників, які з ними будуть інтегруватись.

Тому результатом огляду є список основних характеристик розглянутих систем, які необхідно перенести в систему віддаленого планування задач.

Основні характеристики розглянутих систем-аналогів:

- використання взаємодії сервер-сервер;
- перегляд стану запланованих оповіщень;
- перегляд статистики відправок;
- можливість налаштування особистого аканту та деталей інтеграції власної системи;
- умовно-безкоштовна модель оплати;
- вичерпна онлайн документація для взаємодії з сервісом;
- перевірка валідності оповіщень за хеш-сумою;

2. РОЗРОБКА ВИМОГ ДО ПРОЕКТУ

На основі проведеного огляду існуючих рішень необхідно розробити детальні вимоги до реалізації системи віддаленого планування оповіщень. Розробка вимог буде проведена у два етапи: визначення з технічних вимог до системи і визначення функціональних вимог до системи.

2.1 Технічні вимоги до системи

Оскільки система повинна бути одночасно інтегрованою з великою кількістю клієнтських програмних засобів, що тягне за собою велике навантаження на сервери, на яких буде розгорнута система, тому система повинна спроектована та запрограмована з можливістю швидкого горизонтального масштабування [8].

Також, варто зазначити, що система буде розвиватись та доповнюватись новими типами оповіщень, тому наступною вимогою є модульність системи, яка дозволить без зусиль додати новий тип оповіщень як окремий модуль швидко і легко.

Наступною вимогою є використання моделі детального і комплексного логування помилок, які будуть відбуватись у системі. В першу чергу це необхідно для налагодження та знаходження помилок, адже левова частка роботи системи буде відбуватись без видимого результату для користувача до моменту запізнення запланованого оповіщення або ж іще гірше — його відсутності.

Детальний список рівнів небезпеки для лог-записів, які будуть створюватись системою:

- інформаційний — рівень логування, що має використовуватись під час роботи найскладніших компонентів системи, щоб завжди мати інформацію про їх стан;

- налагоджувальний — даний рівень повинен використовуватись для записів, які створені внаслідок процесу налагодження;
- попереджувальний — цей рівень повинен використовуватись для логування помилок, які викликані неправильними діями користувача та є очікуваними системою;
- критичний — цей рівень повинен використовуватись у разі, якщо внаслідок дій користувача сталась, неочікувана розробниками, помилка або ж помилка була перехоплена у процесі відправки запланованого оповіщення;

Однією із технічних вимог також є використання унікальних ідентифікаторів запитів, які будуть повертатись користувачам у випадах помилок та логуватись, включаючи увесь контекст запиту до системи. Це дасть користувачам можливість сповістити розробників системи про проблему та відразу побачити всю необхідну інформації, щоб її відтворити.

Вимогою до рівня безпеки системи є використання сучасних, криптостійких алгоритмів хешування для користувацьких паролів, а також включення IP-адреси клієнтської системи до процесу авторизації.

Також до рівня безпеки відноситься вимога про приховання у HTTP-відповідях усіх чутливих даних системи: стеки викликів, секретні ключі та простори імен типів, адже з їх допомогою зловмисники можуть дізнатись про базову архітектуру системи та почати проводити на неї атаки з метою викрадення особистих даних користувачів.

Останньою технічною вимогою є детальна онлайн документація, що повинна генеруватись автоматично після кожної зміни в публічному інтерфейсі системи, з яким будуть працюватись розробники.

2.2 Функціональні вимоги

Вимоги до користувацьких можливостей з налаштування облікового запису та інтеграції системи:

- можливість реєстрації у системі, з введенням:
  - імені;
  - прізвища;
  - логіну;
  - паролю;
- можливість входу в систему за допомогою логіну та паролю;
- можливість оновлення власного профілю;
- можливість зміни паролю;
- можливість перегляду та зміни серверного ключа;
- можливість налаштування списку дозволених для авторизації IP-адрес;

Вимоги до процесу серверної авторизації з системами, що будуть використовувати планувальник оповіщень:

- процес авторизації повинен здійснюватись за допомогою стандартизованого HTTP-хедера «Authorization» [9] із схемою «ApiKey»;
- серверний ключ повинен генеруватись автоматично після створення користувацького профілю;
- серверний ключ повинен бути створений за допомогою криптографічно стійкого генератора псевдовипадкових чисел, довжина ключа не менше 128 байт;
- під час перевірки серверного ключа, потрібно враховувати список дозволених для доступу до нього IP-адрес;
- з одним серверним ключем потрібно мати змогу авторизуватись паралельно;

Загальні вимоги до планувальника, які є спільними для HTTP та Email оповіщень:

- можливість змінювати деталі вже запланованого оповіщення;
- можливість запланувати відправку оповіщення за допомогою CRON-виразів [10];
- можливість відміняти відправку запланованих оповіщень;
- можливість переглядати список запланованих оповіщень;
- можливість переглядати список оповіщень, що уже відправлено;
- можливість переглядати список оповіщень, які не змогли відправитись;
- можливість переглядати усі виконання відправок запланованого оповіщення;
- можливість отримання статистики відправок оповіщень;

Функціональні можливості процесу планування HTTP-оповіщень:

- можливість використовувати для створення та відправки оповіщення усі HTTP-методи [2];
- можливість встановлювати HTTP-хедери оповіщення;
- можливість встановлювати контент оповіщення;
- можливість переглядати HTTP статус-код, який отримала система у відповідь на відправку оповіщення;
- можливість змінювати домен адресата запланованого повідомлення;

Функціональні можливості процесу планування Email-оповіщень:

- можливість використовувати HTML шаблонізатор для формування тіла електронного листа;
- можливість відправки одного оповіщення багатьом отримувачам;
- можливість встановлення прихованої копії для оповіщення;
- можливість змінювати отримувачів уже запланованого оповіщення;
- можливість додавати змінні, що будуть підставлені у шаблон HTML для кожного листа;

– можливість встановити власний SMTP сервер для відправки оповіщень;

Висновки до розділу 2

У розділі було розроблено перелік технічних та функціональних вимог до системи, що розглядається у даному дипломному проекті. Основою для цих вимог були вихідні дані для роботи та огляд існуючих рішень.

Серед сформованих вимог найважливішими є вимоги до безпеки застосунку, загального процесу планування, а також технічні вимоги, що будуть впливати на відображення результатів роботи системи кінцевим користувачам.

Варто відзначити, що розроблені технічні та функціональні вимоги не включають деталей реалізації, алгоритмів роботи системи, а також технологій, що повинні бути використаними, оскільки вони повинні бути розроблені на основі вимог викладених у даному розділі.

3. ОГЛЯД ТЕХНОЛОГІЙ ТА ЗАГАЛЬНОПРИЙНЯТИХ ПРИНЦИПІВ СТВОРЕННЯ WEB-ЗАСТОСУНКІВ

На основі висунутих вимог необхідно провести огляд обраних для використання технологій та принципів побудови WEB-застосунків, а також обґрунтувати їх вибір для подальшої розробки.

3.1 Мікросервісна архітектура

Оскільки система, що розглядається в рамках даного дипломного проекту, повинна бути модульною та зручною для горизонтального масштабування [8], то найкращим рішенням для побудови її архітектури є мікросервісна архітектура [11-12].

В основі даного типу архітектури застосунків лежить композиція незалежних модулів — сервісів, кожен з яких вирішує конкретну бізнес-задачу та архітектурно є слабкозв’язаним із іншими сервісами, що забезпечує безперервну роботу системи у випадку поломки одного з сервісів.

Кожен сервіс у свою чергу може бути розгорнутий декілька разів та формувати групу або ж кластер екземплярів конкретного типу сервісу. Саме такий принцип ідеально підходить для викладених технічних вимог до системи, адже це дозволить швидко і легко масштабувати конкретні типи сервісів або ж і весь застосунок, а у випадку додавання нового типу оповіщень буде достатньо просто розгорнути екземпляри нового сервісу.

Також варто відзначити, що такий підхід у порівнянні із монолітним підходом [13] є набагато більш затратним з точки зору ресурсів та об’єму роботи, адже замість одного процесу, всередині якого працює вся система, у розробників є велика кількість таких процесів, які необхідно супроводжувати та контролювати.

Але найбільшою проблемою такого підходу є процес розгортання екземплярів сервісів, адже для кожного типу сервісу необхідні різні технології та залежності, саме цю проблему вирішує технологія контейнеризації Docker.

3.1.1 Технологія контейнеризації Docker

В основі даної технології лежить поняття контейнеризації, що є одним з методів віртуалізації [15], при якому ядро операційної системи підтримує кілька ізольованих та незалежних користувацьких просторів, які і називаються контейнерами. Це дозволяє розгортати в рамках однієї віртуальної машини декілька незалежних та ізольованих контейнерів із застосунками.

Технологія Docker дозволяє не тільки створювати контейнери, але й будувати образи застосунків [14]. Образом Docker є легковісний дистрибутив операційної системи Linux, на який встановлюються всі необхідні сервісу залежності і власне сам сервіс, після чого він запечатується і стає доступним тільки для копіювання та розгортання контейнера на його основі.

Також варто розглянути основні функціональні можливості Docker:

- можливість керування життєвим циклом контейнерів, що дозволить автоматично перезапускати екземпляр сервісу у випадку критичної помилки;
- можливість підключення контейнерів до спільної із віртуальною машиною мережі, що дозволить надати доступ до контейнера з-за меж серверу;
- можливість обмежити ресурси сервера, які використовує контейнер;
- можливість виділення виділеної контейнеру пам’яті на жорсткому диску віртуальної машини, для збереження постійних даних;

Головною перевагою такого підходу є абсолютно незалежний від середовища процес розгортання, який гарантує однакову поведінку та працездатність застосунку на будь-якому сервері, де він буде розгорнуто контейнери з його сервісами.

3.1.2 Брокер повідомлень RabbitMq

Оскільки система буде побудована на основі мікросервісної архітектури, необхідно обрати засіб, який буде забезпечувати стабільний та швидкий процес обміну повідомленнями між екземплярами сервісів. Таким засобом є брокер повідомлень RabbitMq [16-17].

Він являє собою систему для передачі повідомлень, яка організована у вигляді черг, тобто RabbitMq надає гарантію послідовної доставки повідомлень між сервісами, навіть за умови конкурентного використання черги.

В основі роботи RabbitMq лежить обгортка над TCP протоколом — протокол AMQP версії 0.9.1 [17], який оперує наступними термінами:

- споживач — той хто отримує повідомлення з черги RabbitMq;
- виробник — той хто відправляє повідомлення RabbitMq;
- обмінник — сутність протоколу, яка відповідає за визначення черги або кількох черг, в яку повинно потрапити повідомлення від виробника;

Кожне повідомлення по протоколу може складатись із двох частин: тіла, яке може містити будь-який контент, і заголовків, які можуть містити інформацію для ідентифікації типу повідомлення. Також повідомленням можна задати час, через який, у випадку їх не прочитання, вони будуть автоматично видалень або ж навпаки записані на жорсткий диск.

Надзвичайно важливим фактором вибору цього засобу обміну повідомленнями є його висока швидкодія — більше одного мільйона

повідомлень на секунду [18], що забезпечується широкими можливостями кластеризації та масштабування.

3.1.3 Принцип CQRS

Одним з найважливіших принципів, які будуть використані у системі, є принцип розділення відповідальності між запитами за отриманням даних та запитами на зміну стану системи — CQRS або ж «Command Query Responsibility Segregation» [19].

Це відносно новий архітектурний принцип організації роботи системи, який вимагає чіткого розділення процесів запису та читання у системі, тобто кожна операція, що вносить будь-які зміни до системи, зобов’язана бути відокремленою від операції, яка може повернути результат цих змін, а також сама не повинна повертати змінені дані. Відповідно і операція, яка має вичитати та повернути дані, не може вносити зміни до стану системи. Абстрактна структурна схема систем побудованих за даним принципом зображена на рисунку 3.1.

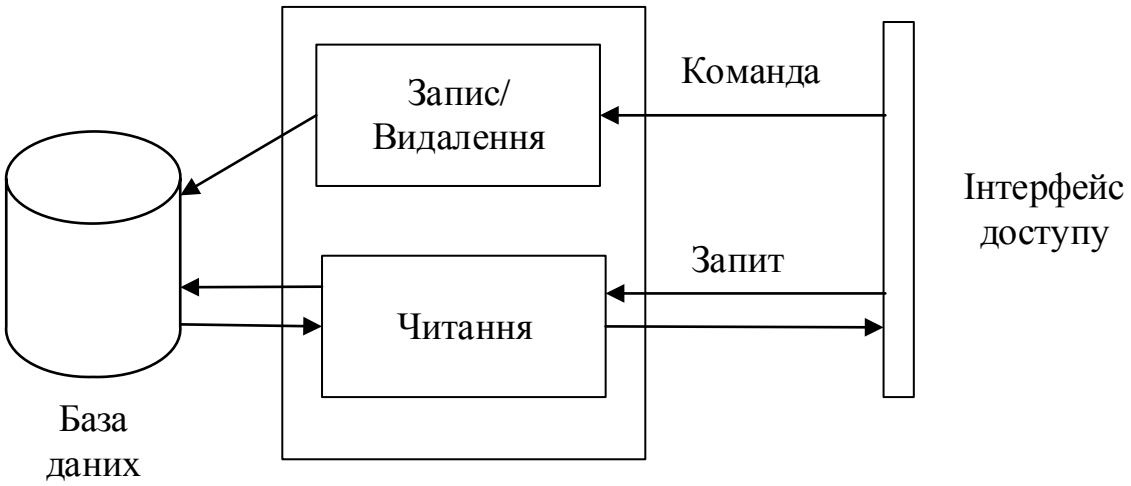

Рисунок 3.1 — абстрактна архітектурна схема системи з використанням CQRS

На основі рисунку 3.1 необхідно розглянути три можливі типи повідомлень, які можна використовувати в рамках принципу CQRS:

- команда — повідомлення, яке викликає зміни в стані системи;
- запит — повідомлення, на яке повинна бути отримана відповідь з інформацією про стан системи;
- подія — повідомлення, яке з’являється внаслідок виконання команди. Обробник повідомлення події також може вносити зміни до стану системи або ж використовувати для цього відповідну команду;

Через наявність чітких правил та вимог даний принцип дуже позитивно впливає на якість написаного коду та його майбутню підтримку, адже при їх дотриманні з’являється чітка структура доменної логіки на рівні абстрактних повідомлень, за якими можна відслідкувати усі місця реалізації.

Саме такий принцип і чітке розділення відповідальності надзвичайно добре підходять для взаємодії між сервісами в контексті мікросервісної архітектури, а також для роботи із базою даних замість застарілих репозиторіїв.

3.2 Система управління базами даних

Робота з даними у системі, що розглядається, є надзвичайно важливим аспектом, тому спочатку визначити тип СУБД [20], яку необхідно використовувати після чого знайти конкретне рішення, яке найкраще підходить для системи планування оповіщень.

Оскільки система повинна буде зберігати денормалізовані дані з контентом оповіщень, варто звернути увагу на об’єктно-реляційні та документо-орієнтовані СУБД.

Документо-орієнтовані бази даних дають змогу зберігати дані будь-якої структури у вигляді документу, що містить в собі всі атрибути об’єкту,

який він зберігає. Даний підхід дуже зручний у випадку використання даних, структура, яких може бути змінною, але такий принцип не гарантує збереження структури документу, що може призвести до непердбачуваних ситуацій під час роботи системи, яка використовує таку базу даних. Тому кращим вибором буде о'єктно-реляційна система управління базами даних, яка представляє дані у вигляді таблиць з чіткою структурою, описом назв та типів колонок, серед яких можуть бути типи, які дозволяють зберігати неструктуровані дані, наприклад XML або ж JSON. Робота з такими СУБД відбувається за допомогою мови структурованих запитів SQL [21].

Серед сучасних об'єктно-реляційних систем управління базами даних обрано PostgreSQL, адже вона є найбільш розвиненою серед безкоштовних аналогів [22].

Головними функціональними можливостями СУБД PostgreSQL, що будуть використані під час розробки є:

- підтримка шести типів індексів [23], що дозволить зробити складні вибірки даних значно швидшими;
- мова Pl/pgSQL, яка є поєднанням процедурного стилю програмування та мови SQL;
- можливість написання функцій на Python та C;
- робота із колонками типу JSON, що дасть зможу зберігати контент оповіщень, без необхідності використання чіткої структури типів;

Також варто зазначити, що PostgreSQL має велику кількість засобів для масштабування баз даних на серверах, та реплікації даних між ними.

3.3 Принципи SOLID

Для вирішення абсолютної більшості задач, з якими доводиться стикатись сучасним розробникам, вже існують готові шаблони їх вирішення та загальноприйняті правила їх реалізації.

«SOLID» — це акронім, який являє собою набір практик проектування структури програмного коду та побудови гнучкої та стійкої до змін системи [24].

Single Responsibility Principle — принцип єдиного обов'язку. Суть цього принципу полягає в тому, що клас повинен виконувати одну єдину, відокремлену логічно, задачу. Конкретне застосування цього принципу залежить від контексту, але в цілому його можна застосовувати до класів, які виконують кілька незалежних функцій в системі.

Open / Closed Principle — принцип відкритості / закритості. Суть цього принципу полягає в тому, що всі сутності в програмному коді, у випадку додання нового функціоналу, повинні лише розширюватись, а не змінюватись, що дозволить не повторювати процес тестування один і тих самих класів чи структур, які частково змінились.

Liskov Substitution Principle — принцип підстановки Лісков. Він полягає у тому, що класи-нащадки повинні наслідувати поведінку батьківських класів так, щоб у випадку заміни одного нащадка на іншого, в місцях виклику де очікується отримати батьківський клас, програма продовжила свою роботу, без будь-яких змін.

Interface Segregation Principle — принцип поділу інтерфейсів, полягає в тому, що кожен інтерфейс системи, повинен мати мінімально можливу кількість функціоналу. Використовується для уникнення випадків, коли класи в програмі реалізовують надлишковий інтерфейс. Тобто велика кількість властивостей або методів класу не використовується або ж взагалі не була реалізована у цьому класі.

Dependency Inversion Principle — принцип інверсії залежностей. Він забороняє використання явних посилань на реалізації, натомість вимагає посилатись на абстрактні сутності системи: інтерфейси, абстрактні класи та фабрики. Це дозволяє розробникам писати слабкозв'язаний код, який буде

стійким до можливих змін, а також легким в тестуванні, адже в посилання на інтерфейс можна завжди передати клас-заглушку.

Дотримання цих принципів не є обов'язковим та нерідко буває надлишковим, але для системи, що розглядається, дасть широкі можливості для розширень та покращень.

3.4 Платформа .NET Core

.NET Core — платформа для розгортання та розробки програмних продуктів, з відкритим вихідним кодом, що розробляється та підтримується компанією «Microsoft». Вона є спадкоємцем платформи .NET, але на відміну від попередника, може працювати на більшості сучасних операційних систем, таких як: Windows, Linux та MacOS. .NET Core дає можливість використовувати в одному програмному продукті три мови програмування: Visual Basic, F# та C#.

Visual Basic — спадкоємець мови BASIC, що дозволяє навчитися розробляти різноманітні програми на .NET Core. Серед мов .NET синтаксис VB найкраще відповідає звичайному синтаксису природної мови, часто спрощуючи розробку для новачків.

F# — функціонально-імперативна мова, що також підтримує використання парадигм ООП. Характеризується складністю початкового входження та використанням конструкцій і принципів, що часто використовуються в найпопулярніших мовах функціонального програмування.

C# — високорівнева мова програмування, що повністю базується на використанні парадигм ООП, має систему статичної та строгої типізації [27], що забезпечує повну безпеку використання типів. Синтаксично вона є спадкоємцем C++, при цьому вносить багато нових синтаксичних

конструкцій, таких як: лямбда-вирази, анонімні типи, асинхронні методи та події [26].

Сама платформа .NET Core складається з двох частин: набору інструментів та бібліотек для розробників — CoreFX і CoreCLR — середовища для виконання скомпільованого .NET коду. Для повного розуміння того, як система, що розглядається у даному бакалаврському проекті, буде розгортатись та розроблятись потрібно розглянути обидві частини .NET Core детальніше.

CoreFX надає величезну кількість готових бібліотек для роботи з колекціями, файлами, мережею, базами даних та ін.., що розповсюджуються у вигляді незалежних пакетів. Це дозволяє запаковувати їх в один архів з застосунком перед розгортанням, без безпосереднього їх встановлення на сервері розгортання. Також варто зазначити, що ці бібліотеки активно розвиваються і підтримуються розробниками, які постійно включають виправлені помилки в нові релізи. Зазвичай, нова патч-версія CoreFX виходить щомісяця.

CoreCLR в свою чергу є головною частиною платформи .NET Core, адже це саме вона робить виконання коду, написаного на мовах програмування .NET, безпечним та можливим до виконання на різних операційних система з різною архітектурою процесора [25]. Тобто саме через CoreCLR, розробники на C# чи F# не можуть просто так отримати можливість прямого читання з оперативної пам’яті або ж прямої взаємодії із операційною системою. CoreCLR також відповідальна за очистку ресурсів, які використовують розробники, тим самим даючи їм змогу не витрачати час на збирання використаної пам’яті [25]. Варто зазначити, що через можливість роботи CoreCLR на операційних системах Linux, вона добре підходить для роботи з технологією Docker, що була описана у підрозділі 3.1.1.

3.5 Бібліотеки для роботи з базами даних

Оскільки для роботи з сучасними базами даних необхідно працювати з мережею по протоколу TCP, задля зменшення ризиків помилок з боку розробників існують готові бібліотеки для взаємодії з СУБД. Для PostgreSQL це бібліотека NpgSQL (.NET PostgreSQL), що забезпечує можливість використання усіх функцій СУБД у застосунках, які її використовують.

3.5.1 Entity Framework Core

Так як представлення даних у вигляді таблиць не відповідає представленню даних у вигляді зв’язаних об’єктів, розробники використовують ORM-системи. Це засоби, які дозволяють автоматично привести результат вибірки з бази даних у відповідний йому об’єкт або ж цілу ієрархію. Найпопулярнішою в середовищі .NET Core ORM-системою є Entity Framework Core, далі EF.

EF надає розробникам іще один рівень абстракції, який дозволяє абстрагуватися від самої бази даних і працювати з даними незалежно від типу сховища. Якщо на рівні бази даних розробники оперують таблицями, індексами, первинними і зовнішніми ключами, але на концептуальному рівні, який пропонує EF, робота відбувається з об’єктами. Центральною концепцією EF є поняття сутності або entity. Сутність представляє набір даних, асоційованих з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх колекціями. Це досягається використанням типу Expression з середовища .NET Core. Кожен Expression, який застосовується до об’єкта контексту бази даних, транслюється в SQL-запит, виконується після чого результат запиту знову перетворюється в .NET об’єкт.

Також EF надає інструментарій для контролю схеми бази даних, що називається міграціями. Міграція в EF — це SQL-скрипт, який переносить усі сутності, які мають використовуватись, в схему бази даних, створюючи необхідні таблиці, індекси, вторинні ключі та функції. Ця можливість гарантує, що усім сутностям, які використовуються застосунком, існує відповідна таблиця в базі даних. Варто зазначити, що розробник може сам впливати на створення скриптів міграцій, налаштовуючи контекст вручну.

3.5.2 Dapper

Трансляція виразів, написаних на C#, у відповідний провайдеру діалект SQL, бібліотеці EF вдається досить успішно, але проблеми починаються в момент, коли вимоги до бази даних зростають і стає необхідним виконувати складні запити, які неможливо написати за допомогою LINQ. Для таких випадків існує лекговісна і надшвидка [28] ORM-система Dapper, що дозволяє виконувати запити за допомогою сирого SQL, та отримувати результат у вигляді .NET об’єктів.

Секретом швидкості роботи цієї бібліотеки є безпосереднє використання провайдерів ADO .NET, які є частиною CoreFX, а відповідно оптимізовані розробниками самої платформи, і, звичайно, процес кешування збудованих функцій відповідності SQL-запиту до типу C# об’єкта.

Поєднання EF та Dapper дає розробникам зручність, у роботі з базою даних та швидкість виконання запитів у критичних місцях.

3.6 Autofac

Для повного розуміння призначення цієї технології, спочатку необхідно розглянути базові для її використання терміни.

Inversion of Control (інверсія управління) — це набір рекомендацій для написання слабкозв'язаного коду. Суть якого полягає в тому, що кожен компонент системи повинен бути якомога більш ізольованим від інших, не покладаючись в своїй роботі на деталі конкретної реалізації інших компонентів.

Dependency Injection (впровадження залежностей) — одна з реалізацій цього принципу, в основі якої лежить використання додаткового об’єкта, який відповідальний за впровадження в інший об’єкт усіх його залежностей.

IoC-контейнер — це бібліотека, що дозволяє автоматизувати використання реалізацій принципу інверсії управління, без явної роботи з залежностями.

Однією з таких бібліотек є саме Autofac. Він дозволяє не лише вирішувати проблеми залежностей об’єктів, а також керувати часом їх життя та областями застосування.

Бібліотека Autofac має наступні типи областей застосування для об’єктів, що будуть створені контейнером:

- Instance Per Dependency — об’єкт буде створюватись щоразу, коли він необхідний, для впровадження залежності;
- Single Instance — об’єкт буде створено лише один раз за період роботи застосунку;
- Instance Per Thread — об’єкт буде створений один раз для кожного потоку, якому він буде необхідний для впровадження залежності;
- Instance Per Request — об’єкт буде створюватись для кожного HTTP-запиту лише один раз;

Оскільки для написання системи буде використано технологію ASP .Net Core, варто зазначити, що для неї IoC-контейнер Autofac має вбудовану інтеграцію.

3.7 Бібліотека для планування задач Quartz .Net

Основою для планування оповіщень у системі стане бібліотека з відкритим вихідним кодом Quartz .Net. Вона є портованою на платформу .NET Core версією однойменного аналога зі світу мови програмування Java.

Робота з цією бібліотекою супроводжується постійним використанням двох понять: job (задача) і trigger (тригер).

Job — це сутність, яка описує те що буде заплановано та виконано за графіком, проте про сам графік виконання вона має інформації. Кожна задача має свій унікальний ідентифікатор, який складається з її назви та назви групи задач. Даний підхід дозволяє дуже зручно керувати різними типами задач, що повинні виконуватись за графіком.

Trigger — це сутність, яка описує правила планування задачі: дату її початку, кількість повторень, інтервал між повтореннями і стратегію виконання у випадку помилки. Тригери також мають унікальні ідентифікатори, але можуть використовуватись для багатьох задач одночасно. У випадку, якщо тригер використовується тільки для однієї задачі, яку відміняють у системі, то відміняється робота самого тригера.

Варто також відзначити підтримку CRON [10] тригерів, які необхідні для використання у системі, що розглядається, на основі висунутих вимог з розділу 2. Бібліотека Quartz .Net для їх виконання використовує системні календарі та інформацію про зміну дня та ночі. Приклад CRON-виразу: 0 0 12 * * SUN,SAT, означає, що задача буде виконана кожної суботи та неділі опівдні.

Для процесу планування задач за графік, дуже важливою частиною є збереження задач та тригерів, які будуть виконані. Quartz .Net надає публічний інтерфейс, який можна реалізувати з використанням будь-якого зручного сховища даних, проте сама бібліотека надає наступні, вже реалізовані, провайдери сховищ:

– сховище в оперативній пам’яті застосунку;
– сховище в NoSQL базі даних MongoDb;
– сховище в SQL базах даних;

Оскільки система буде використовувати СУБД PostgreSQL для роботи з даними, то і для збереження запланованих задач доцільно використовувати таку ж СУБД, адже сховище організоване в оперативній пам’яті неможливо буде відновити у випадку рестарту системи.

Використання зовнішнього сховища даних також дозволить задовольнити вимоги мікросервісної архітектури, яка вимагає відсутності збереження стану системи в оперативній пам’яті, що відкриває шлях до розгортання цілого кластеру планувальників, які будуть синхронізовані на рівні бази даних за рахунок окремої таблиці з блокуваннями.

3.8 Засоби автоматичного створення онлайн документації

Онлайн документація є дуже важливою складовою будь-якого публічного API, оскільки саме завдяки ній, розробники, що з ним інтегруються, дізнаються про процес взаємодії із системою, отримають опис та приклади запитів і проводять початкове тестування правильності інтеграції.

Така документація повинна відповідати стандарту OpenApi [30]. Він виставляє наступні загальні вимоги для самого публічного API та його документації:

– формат згенерованої документації має бути JSON або YAML, тому, що ці формати легкі для читання як людино так і машиною;
– вона повинна описувати всі доступні шляхи для HTTP запитів;
– документація повинна описувати всі параметри, які повинен мати кожен HTTP-запит;
– процес авторизації також повинен бути включений в документацію;

Для дотримання даного стандарту і генерації автоматичної документації з веб-інтерфейсом обрано бібліотеку Swashbucke.AspNetCore, яка використовує вбудований в ASP .NET Core, провайдер REST API інтерфейсу. Приклад згенерованої сторінки документації зображено на рисунку 3.2.

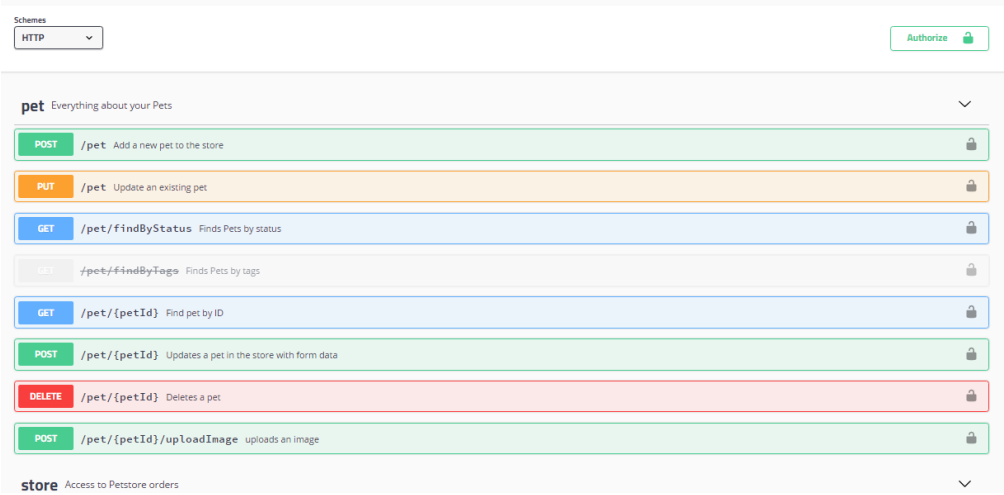

Рисунок 3.2 — Онлайн документація, згенерована засобами Swashbucke.AspNetCore

3.9 HTML шаблонізатор Handlebars

Відповідно до вимог викладених в розділі 2, система повинна надавати можливість шаблонізації контенту електронних листів, для таких цілей використовуються HTML-шаблони, які відправляються в тілі повідомлення та відображаються браузерами або застосунками як вбудований у сторінку HTML.

Серед шаблоні заторів HTML один з найпопулярніших є Handlebars, який компілює HTML разом із вставками JavaScript коду, який відповідальний за підстановку даних моделі в шаблон.

Основні операції доступні для використання у шаблонах:

– робота з JSON об’єктами, отримання їх властивостей по шляху;

– використання циклів for та foreach;
– повноцінна підтримка масивів, можливість отримання значення за індексом;
– можливість використання коментарів;

Приклад HTML шаблону Handlebars наведено на рисунку 3.3, в основі шаблону лежить звичайна HTML розмітка, в яку під час компіляції підставляються змінні, з такими самими назвами як і у шаблоні.

Для роботи з масивами об’єктів визначений оператор #each, який перелічує кожен елемент масиву. Якщо елементом є об’єкт, то оператор #each дає можливість звертатись до його полів напряму.

```
<div class="post">
  <h1>By {{fullName author}}</h1>
  <div class="body">{{body}}</div>

  <h1>Comments</h1>

  {{#each comments}}
  <h2>By {{fullName author}}</h2>
  <div class="body">{{body}}</div>
  {{/each}}
</div>
```

Рисунок 3.3 — Приклад шаблону Handlebars

Висновки до розділу 3

В рамках цього розділу було оглянуто технології та принципи побудови сучасних WEB-застосунків, які дають можливість реалізувати всі висунуті технічні та функціональні вимоги.

Мікросервісна архітектура дозволить системі бути модульною та масштабованою, а технологія Docker та платформа .NET Core, дозволить використовувати операційні системи Linux, що значно здешевить процес підтримки системи після розробки.

Принцип CQRS та правила SOLID забезпечать надзвичайно легке впровадження додаткових модулів із новими типами оповіщень у систему. Оскільки дані принципи досить широко застосовуються у сучасній розробці, то їх використання не стане проблемою для людей, які будуть займатись розробкою нових модулів для системи.

Розглянуті засоби роботи з базою даних, дозволять працювати з нею і швидко, і зручно, що буде досягнуто використанням одразу двох різнопланових бібліотек.

Усі оглянуті технології є абсолютно безкоштовними та можуть використовуватись для комерційної та некомерційної розробки застосунків.

Технології та принципи, що були згадані вище, разом забезпечать необхідне технічне ядро системи, яке стане основою для реалізації бізнес-задач, викладених у функціональних вимогах до системи.

4. РОЗРОБКА СИСТЕМИ

На основі, висунутих у розділі 2, вимог до системи та огляду технологій і принципів проектування, що повинні бути в ній застосовані, розробка системи була розбита на великі комплексні задачі, які виконувались послідовно.

Спочатку необхідно оглянути розроблену структурну схему системи з додатку А. Уся система складається з семи структурних частин: балансувальник навантаження, публічний сервіс API, сервіс планування задач, сервіс виконання HTTP запитів, сервіс відправки Email оповіщень, шина даних RabbitMQ та база даних.

Між API сервісом та зовнішнім світом знаходиться балансувальник навантаження, який забезпечить ізоляцію екземплярів API сервісу від зовнішнього світу на рівні мережі, а також може розподілити навантаження між екземплярами на основі нормального розподілу запитів.

Взаємодія між сервісами системи відбувається виключно через шину повідомлень RabbitMQ. Варто зауважити, що всі сервіси є рівнозначними у системі і кожен з них може спілкуватись з будь-яким іншим без обмежень, але лише через шину повідомлень.

Сервіс публічного доступу до системи, або ж просто API. Він відповідальний за процес авторизації, налаштування користувацьких інтеграцій та безпосередній доступ інтегрованих систем. Це єдиний сервіс, до якого є доступ із-за меж кластеру системи.

API сервіс, використовує базу даних для збереження даних про інтегровані системи, авторизацію та оповіщення і всі залежні від них сутності.

Сервіс планування задач відповідальний за планування усіх типів задач, що будуть використані в системі. Використовує базу даних для

збереження стану планування кожної задачі та для синхронізації із іншими екземплярами планувальника.

Сервіс відправки HTTP оповіщень відповідальний за їх безпосередню відправку, обробку результатів, обробку виключних ситуацій при відправці.

Сервіс відправки Email оповіщень відповідальний за перевірку на правильність шаблонів оповіщень, підстановку параметрів у шаблони, компіляцію шаблонів і безпосередню відправку повідомлень з обробкою можливих помилок.

Кожен з сервісів використовує зовнішні конфігурації, за допомогою яких можна змінювати його поведінку на рівні розгортання застосунків. Наприклад, якщо потрібно змінити сервер бази даних, оновити авторизаційні ключі або ж змінити місце куди сервіс пише логи. При старті або перезавантаженні кожного з сервісів, йому обов'язково передається посилання для завантаження зовнішнього JSON файлу, в якому безпосередньо описана його конфігурація. Приклад docker-compose файлу для розгортання одного з сервісів зображено на рисунку 4.1.

```
version: '3'

services:
  api:
    image: ${CI_REGISTRY_IMAGE}/api:${TAG_VERSION}
    container_name: remote-scheduler-api-${TAG_VERSION}
    build:
      context: .
      dockerfile: RemoteScheduler.Api.Dockerfile
    environment:
      - "HOSTING_ENVIRONMENT=${HOSTING_ENVIRONMENT}"
      - "APPSETTINGS_URL=${APPSETTINGS_URL}"
      - "LOG4NET_URL=${LOG4NET_URL}"
    ports:
      - "127.0.0.1:6000:80"
```

Рисунок 4.1 — Приклад docker-compose файлу для розгортання сервісу API

Також на структурній схемі системи бази даних та шина повідомлень зображені як такі ж її компоненти, які необхідно розгортати та супроводжувати. На рисунку 4.2 зображено налаштування docker-compose файлу для розгортання PostgreSQL та RabbitMQ.

```
rabbit-mq:
    image: "rabbitmq:3-management"
    container_name: remote-scheduler-rabbit
    environment:
      RABBITMQ_ERLANG_COOKIE: "SWQOKODSQALRPCLNMEQG"
      RABBITMQ_DEFAULT_USER: "${RMQ_USER}"
      RABBITMQ_DEFAULT_PASS: "${RMQ_PASS}"
      RABBITMQ_DEFAULT_VHOST: "${RMQ_VHOST}"
    ports:
      - "127.0.0.1:15672:15672"
      - "127.0.0.1:5672:5672"

  postgres:
    image: "postgres:10.7-alpine"
    container_name: remote-scheduler-postgres
    environment:
      POSTGRES_USER: "${PG_USER}"
      POSTGRES_PASSWORD: "${PG_PASS}"
    volumes:
      - /var/lib/docker/volumes/remote-scheduler/postgresql:/var/lib/postgresql/data
    ports:
      - "127.0.0.1:5432:5432"
```

Рисунок 4.2 — Приклад docker-compose файлу для розгортання технічних сервісів системи

Для спільних технічних рішень виділений простір імен Core, який містить у собі реалізацію CQRS, системи конфігурацій, логування та моделі виключень.

Структура коду розробленого проекту зображена на рисунку 4.3. Кожен сервіс складається з чотирьох основних збірок, які дозволяють чітко описати структуру кожного з них:

– збірка з публічними контрактами сервісу, за допомогою яких інші сервіси будуть взаємодіяти з ним;
– збірка з абстракціями цього сервісу, які доступні тільки йому і містять в собі основні інтерфейси, з якими працює цей сервіс;

– збірка з реалізацією абстракцій, в якій реалізовані обробники команд та запитів із публічних контрактів, а також реалізовані інтерфейси з абстракцій;
– консольний застосунок, що є місцем старту сервісу, вона ініціалізує Autofac контейнер разом з усіма залежностями із збірки з реалізаціями;

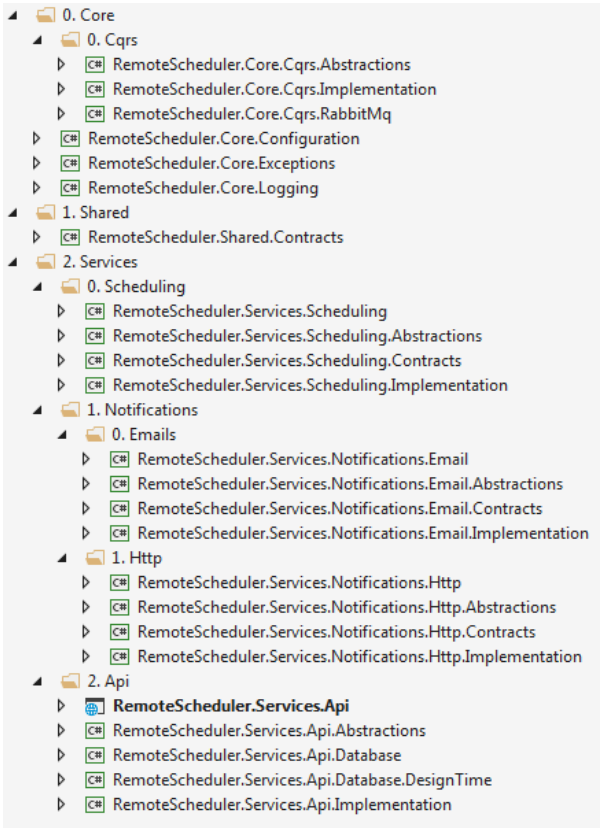

Рисунок 4.3 — Структура проекту системи віддаленого планування оповіщень

Також варто оглянути блок-схему загального алгоритму роботи системи, наведеного у додатку Б.

Блок 1 — записати у змінну HttpRequest вхідний HTTP запит.

Блок 2 — авторизувати HttpRequest.

Блок 3 — якщо HttpRequest не авторизовано, то закінчити виконання з помилкою, інакше — продовжити виконання.

Блок 4 — якщо HttpRequest це не команда, то перейти до блоку 5, інакше — до блоку 9.

Блок 5 — дістати запит з тіла HttpRequest.

Блок 6 — визначити обробник для запиту.

Блок 7 — обробити запит записавши результат у змінну Result.

Блок 8 — якщо результат успішний, то повернути його користувачеві, інакше повернути повідомлення про помилку.

Блок 9 — дістати команду з тіла HttpRequest.

Блок 10 — визначити обробник для команди.

Блок 11 — якщо обробник знайдено, перейти до блоку 12, інакше — до блоку 14.

Блок 12 — обробити команду записавши результат у змінну Result.

Блок 13 — якщо результат успішний, то повернути користувачеві підтвердження успішності його дії, інакше повернути повідомлення про помилку.

Блок 14 — передати виконання на інший сервіс, який має обробник цієї команди, задачу записати у змінну Task.

Блок 15 — асинхронно чекати завершення блоку 17.

Блок 16 — отримати обробник команди.

Блок 17 — обробити команду записавши результат поле Result змінної Task.

Блок 18 — записати результат віддаленого виконання команди у змінну Result.

Блок 19 — якщо результат успішний, то повернути користувачеві підтвердження успішності його дії, інакше повернути повідомлення про помилку.

4.1 Реалізація принципу CQRS для роботи системи

Робота над реалізацією принципу CQRS повинна складатись з трьох чистин: розробка абстракцій контрактів даних, шин та обробників, розробка реалізації CQRS в рамках одного сервісу та безпосередня реалізація з використанням RabbitMQ. Розгляд необхідно почати з абстракцій.

Для контрактів даних було визначено чотири основних інтерфейси, які будуть встановлювати обмеження на типи повідомлень та їх обробників у системі. Команди означені інтерфейсом ICommand, події — IEvent, відповіді на запити реалізують IQueryResult, а самі запити — IQuery<TResult>, да TResult це тип відповіді, яку повертає запит, яка обмежений інтерфейсом IQueryResult. Така структура абстракцій основних сутностей CQRS дозволяє на рівні класів контрактів бачити основну бізнес-логіку системи і відслідковувати місця їх викликів та обробки.

Обробники описані інтерфейсами: ICommandHandler<TCommand>, де TCommand це тип команди, для якої цей обробник призначається, IEventSubscriber<TEvent>, де TEvent це тип події, яка буде обролена обробником і IQueryHandler<TQuery, TResult>, де TQuery це тип запиту, а TResult це тип відповіді. Така структура дозволяє дуже легко та швидко знайти відповідний команді чи запиту обробник, відслідкувавши інтерфейс де використовується тип команди чи запиту. У випадку великих систем, із сотнями розробників, така структура дозволить дуже строго стандартизувати роботу з обробкою повідомлень.

Для використання команд, запитів та подій створено інтерфейси шин повідомлень таких типів, призначенням яких є визначення обробника для потрібного типу повідомлення та його виклик. Такими інтерфейсами є: ICommandBus, IQueryBus і IEventBus.

Варто зазначити, що використання інтерфейсів шин дає можливість використовувати у системі принцип Location Transparency або ж прозорості

розміщення. Його суть полягає в тому, що код, який викликає команду, подію чи запит, не повинен знати яким мікросервісом вона буде оброблена. Тобто фактично виклик команди, обробник якої знаходиться в домені того ж сервісу, не відрізняється від виклику команди, обробник якої знаходиться в іншому сервісі, на іншій ноді в кластері системи.

Реалізація шин повідомлень в рамках одного сервісу, полягає в тому, що шина даних по суті виступає диспетчером, який за типом повідомлення знаходить його обробник, та передає контроль над обробкою повідомленням йому. У випадку команд та запитів такий обробник може існувати один і тільки один, а у випадку подій обробників може бути безліч.

Пошук відповідного обробника, на кожне повідомлення відбувається серед зареєстрованих типів в IoC-контейнері Autofac, з областю застосування Instance Per Dependency, що дозволяє створювати на одне повідомлення ексклюзивний обробник, який знищиться після завершення своєї роботи. При такому підході необхідно слідкувати за тим, щоб обробники повідомлень не інкапсулювали в собі стан, адже в таких випадках необхідно забезпечити синхронізацію роботи конкурентних запитів зі зміною збереженого стану, що приведе до значного зниження продуктивності системи. Тому під час реалізації бізнес-логіки було дотримане правило, що не дозволяє зберігати в обробниках стан.

Реалізація взаємодії між сервісами за допомогою CQRS і RabbitMQ, лише реалізовує необхідні інтерфейси та використовує шину повідомлень в кожному з сервісів для визначення місця існування обробника повідомлення.

Для відправки повідомлень в RabbitMQ, створено обробники на кожен інтерфейс повідомлень, які є generic або ж шаблонними для всіх повідомлень, які реалізують інтерфейси контрактів. У випадку, якщо обробника не існує в домені сервісу, який викликає команду чи запит, то обирається, згаданий вище, шаблонний обробник, який відправляє

повідомлення в RabbitMQ. Дане правило не діє для подій, вони завжди відправляються в RabbitMQ.

Кожен з сервісів після свого старту створює або ж оновлює існуючу чергу команд, запитів та подій, CommandQueue, QueryQuque і EventQueue відповідно, яка є спільною для кожного типу сервісу і повідомлення з неї будуть надсилатись екземплярам по черзі, що забезпечити балансування навантаження.

Оскільки команди та запити, повинні дати відповідь про результат виконання, кожен з екземплярів має свою ексклюзивну чергу відповідей — RpcResponseQueue, куди приходять результати виконання команд чи запитів. Кожне повідомлення відправлене екземпляром місить хедер ReplyTo, який посилається на чергу відповідей саме цього екземпляра сервісу і CorrelationId — унікальний ідентифікатор, який дозволяє визначити на стороні відправника результат виконання якого повідомлення йому повернувся.

Для подій процес значно простіший, відправник надсилає повідомлення про подію і не очікує відповіді, адже подія буде надіслана усім сервісам системи.

У брокері RabbitMQ на старті сервісів визначаються три обмінники до яких потім приєднуються черги:

- обмінник команд, тип Direct, відправляє команду в одну конкретну чергу, яку він визначає, за назвою простору імен команди, який формує RoutingKey;
- обмінник запитів, тип Direct, відправляє команду в одну конкретну чергу, яку він визначає, за назвою простору імен запиту і типу відповіді, які формують RoutingKey;
- обмінник подій, тип Fanout, надсилає подію абсолютно всім чергам, які до нього приєднані;

Результуюча структура черг представлена на рисунку 4.4.

Overview

Name	Features
RemoteScheduler.Services.Api.CommandQueue	D
RemoteScheduler.Services.Api.EventQueue	D
RemoteScheduler.Services.Api.QueryQueue	D
RemoteScheduler.Services.Api.RpcResponseQueue.1	D
RemoteScheduler.Services.Notifications.Email.CommandQueue	D
RemoteScheduler.Services.Notifications.Email.EventQueue	D
RemoteScheduler.Services.Notifications.Email.QueryQueue	D
RemoteScheduler.Services.Notifications.Email.RpcResponseQueue1	D
RemoteScheduler.Services.Notifications.Http.CommandQueue	D
RemoteScheduler.Services.Notifications.Http.EventQueue	D
RemoteScheduler.Services.Notifications.Http.QueryQueue	D
RemoteScheduler.Services.Notifications.Http.RpcResponseQueue.1	D
RemoteScheduler.Services.Scheduling.CommandQueue	D
RemoteScheduler.Services.Scheduling.EventQueue	D
RemoteScheduler.Services.Scheduling.QueryQueue	D
RemoteScheduler.Services.Scheduling.RpcResponseQueue.1	D

Рисунок 4.4 — Структура черг в RabbitMQ

Детальніше слід оглянути розроблену UML діаграму послідовності роботи принципу CQRS з додаку Б.

Послідовність починається з виконання команди або запиту, оскільки для команд, які належать віддаленому сервісу, в рамках сервісу відправника не визначено обробника, то обирається шаблонний.

Шаблонний обробник реєструє майбутню відповідь в черзі відповідей, формує повідомлення з команди або запиту та відправляє його в обмінник команд або запитів відповідно.

Обмінник на основі RoutingKey визначає відповідальну за команду або запит чергу, та передає повідомлення їй на обробку. Черга за принципом RoundRobin відправляє повідомлення на Listener сервісу виконавця. Якщо в даний момент він не доступний, то черга чекає одну хвилину, після чого видаляє повідомлення, адже відповідь у черзі відповідей очікується лише хвилину, після чого виникає виключення про тайм-аут виконання.

Listener виконавця шукає обробник для даної команди чи запиту та передає керування йому, у випадку, якщо такого не знайдено у чергу відповідей надсилається відповідь з відповідною помилкою.

Обробник виконує команду або отримує інформацію для запиту та повертає результат виконання назад до Listener'а. Він надсилає його у чергу відповідей, яка повертає результат сервісу-відправнику та відміняє очікування відповіді.

Також важливою частиною в процесі взаємодії мікросервісів є формат передачі повідомлень. Усі повідомлення, що проходять через RabbitMQ передаються туди у вигляді просто масиву байт. В масив байт повідомлення приводяться за допомогою Strongly-typed JSON серіалізації. Такий підхід до передачі повідомлень, дозволяє використовувати абстракції в контрактах даних, які під час серіалізації запишуться у повідомлення як конкретний тип, а під час десеріалізації відбудуються назад. Приклад серіалізованого повідомлення зображено на рисунку 4.5.


```
{
  "$type": "RemoteScheduler.Services.Scheduling.Contracts.Commands.ScheduleCommandExecutionCommand",
  "JobId": "50f79c20-8b6c-4b3d-89c6-248e6019b68f",
  "UserId": "66c7ed55-2bee-4129-9908-26a1b8a0d2ad",
  "StartAt": "2019-06-25T21:36:47.741",
  "CronSchedule": null,
  "RepeatsCount": 9,
  "Interval": "00:00:15",
  "Command": {
    "$type": "RemoteScheduler.Services.Notifications.Http.Contracts.Commands.SendHttpNotificationCommand",
    "NotificationId": "a65259ac-cbd5-4372-9789-0f5b35c0fb02",
    "Scheme": "string",
    "Domain": "string",
    "Path": "string",
    "ActionType": 1,
    "Body": {

    }
  }
}
```


Рисунок 4.5 — Приклад Strongly-typed JSON серіалізації

Кожен сервіс на старті створює два підключення до RabbitMQ, одне для відправки повідомлень, а інше для їх отримання. Це дозволяє створювати в рамках підключень окремі канали, користуючись ними паралельно, а також

дозволяє запобігти звортньому тиску в контексті одного підключення з боку брокера RabbitMQ. Адже у випадку, якщо через одне підключення одночасно відправляється багато повідомлень, то брокер їх тимчасово затримає, допоки сам не відправить повідомлення в зворотному напрямку.

На кожну чергу відповідей для команд та запитів відкривається окремий канал, який існує увесь час роботи сервісу, так само як і для відправки або отримання команд, запитів чи подій. Клієнт RabbitMQ для .NET Core гарантує відновлення підключення та каналу у випадку непередбачуваних проблем із мережею. Така поведінка вмикається прапорцем AutomaticRecoveryEnabled на фабриці усіх підключень.

4.2 Структура бази даних

Таблиці бази даних відображають усі сутності системи. Кожна сутність зберігає інформацію про дату створення та дату останнього оновлення, що дає можливість швидко знаходити нові записи в базі даних. Також усі сутності мають унікальні ідентифікатори типу Guid, які є первинними ключами у відповідних таблицях. Зображення структури бази даних наведено у додатку В.

Таблиця _EFMigrationHistory, у ній зберігається технічна інформація про всі версії структури бази даних.

Таблиця users є основною таблицею в усій базі даних, решта таблиць завжди явно або через додатковий зв’язок посилаються на неї. Вона містить інформацію про користувача та налаштування інтеграції його системи, які відображені в атрибутах api_key — серверний ключ доступу та allowed_server_ips — масив дозволених IP-адрес для авторизації.

Таблиця user_refresh_tokens містить інформацію для авторизацію користувача за допомогою логіну та пароля, через які він може змінювати персональну інформацію та налаштовувати інтеграцію своєї системи.

Таблиця email_tempaltes містить інформацію про створені користувацькою системою HTML шаблони для Email оповіщень. Варто зазначити, що на рівні бази даних встановлено обмеження на унікальність ключа шаблону для кожного користувача, за допомого створення складеного унікального індекса на атрибутах key та user_id.

Таблиця user_email_settings зберігає інформацію про, налаштовані користувачем, SMTP сервери.

Таблиця email_receivers зберігає інформацію про отримувачів електронних листів. Атрибут data типу jsonb, використовується в якості змінної при компіляції HTML шаблонів для електронних листів.

Таблиця email_notification_receivers є розв’язочною таблицею зв'язку багато до багатьох між email_receivers та email_notification.

Таблиця jobs відображає заплановані задачі, вона є базовою для всіх типів оповіщень у системі та використовується за принципом Table Per Type, тобто на кожен клас з ієрархії, створюється окрема таблиця, але без дублювання атрибутів, які залишаються доступними через явне посилання на батьківську таблицю у вигляді вторинного ключа. Дана таблиця містить всю базову інформацію про графік відправки оповіщень. У випадку, якщо необхідно використовувати CRON тригер, то атрибут cron_schedule несе в собі тіло самого виразу, а якщо звичайний тригер, то кількість ітерацій доступна за атрибутом repeats_count, інтервал — interval, а час початку — start_at.

Таблиця job_executions містить в собі інформацію про відправку оповіщення за графіком. В атрибут execution_status записується статус відправки кожного оповіщення.

Таблиця email_notifications є нащадком таблиці jobs. Вона містить посилання на отримувачів повідомлення, SMTP сервер, який необхідно використати, а також на шаблон, який повинен бути використаний для цього оповіщення.

Таблиця http_notifications є нащадком таблиці jobs. Вона містить інформацію про кінцевого адресата оповіщення, тип HTTP методу і контент оповіщення, що зберігається в тип jsonb. Важливою особливістю структури атрибутів цієї таблиці є те, що, вона зберігає посилання у вигляді кількох тектових атрибутів, які дозволяють значно пришвидшити операції оновлення та пошуку, завдяки зменшенню довжин строк.

Таблиці http_notification_query_parameters і http_notification_headers зберігаються інформацію про query параметри і хедери оповіщення. На кожній з цих таблиць налаштовано унікальний складений індекс по атрибутам key та notification_id, що забезпечить унікальність хедерів та query параметрів в контексті одного HTTP-оповіщення.

Спроектована база даних не є нормалізованою, адже для швидкої роботи із неструктурованими даними було обрано використовувати jsonb типи, а також використовувати дублювання певних атрибутів, для зменшення кількості операцій JOIN, що забезпечить швидкий доступ до них.

4.3 Сервіс API

В основі даного сервісу лежить ASP .NET Core застосунок, що розгорнутий на веб-сервері Kestrel. Організація публічного інтерфейсу виконується на рівні класів-контролерів, де кожен контролер надає інтерфейс для роботи з окремою логічною одиницею в системи, наприклад: HTTP оповіщення або ж шаблони Email оповіщень.

Сервіс відповідальний за три ключові задачі системи: авторизація у системі, збереження й відображення даних про оповіщення від користувацьких систем і передача на виконання їх команд чи запитів іншим сервісам системи.

Процес обробки запиту, який починається з процесу авторизації. Для реалізації безпечного процесу клієнтської авторизації через логін з паролем і

серверної, за допомогою ключа, реалізоване поняття процесорів схем авторизації. У системі визначено інтерфейс IAuthorizationSchemeProcessor та дві його реалізації. Перша реалізовує Bearer схему авторизації із JWT токеном, друга реалізовує серверну авторизації із схемою ApiKey.

Процес серверної авторизації значно простіший, адже саме поняття авторизації між серверами гарантує те, що ключі, які будуть для неї використовуватись, не будуть доступні публічно під час взаємодії з системою, тобто їх не можна буде побачити у відкритому вигляді, наприклад у браузері.

У першу чергу виконується перевірка на наявність авторизаційного ключа в хедері Authorization із схемою ApiKey, де схемою за стандартом цього хедеру є перше слово із його значення. У випадку відсутності хедера чи значення в ньому, запит завершується з помилкою і 401 статус кодом, що означає «Неавторизовано».

Так як авторизаційний ключ має довжину 128 байт та генерується, за вимогами з розділу 2 за допомогою криптографічно стійкого генератора псевдовипадкових чисел, реалізацією якого в середовищі .NET є клас RNGCryptoProvider, кожен ключ є унікальним, то за отриманим з хедеру ключем необхідно знайти користувача в базі даних та отримати список дозволених IP-адрес. У випадку, якщо користувача не знайдено або публічна IP-адреса адресанта не знайдена у списку дозволених, то запит так само закінчується з помилкою 401. Після цієї перевірки, обробка запиту проходить далі.

Авторизація за допомогою логіна та пароля дещо складніша, адже в такому випадку немає гарантій, що токени доступу, видані системою, не потрапили в руки зловмисників, тому після кожного логіну, користувачеві видається токен доступу, який стає невалідним за 12 годин, та токен оновлення токена доступу або ж рефреш токен, який дає змогу оновити невалідний токен доступу.

Також в кукі HTTP відповіді записується згенерований унікальний ідентифікатор пристрою, який дозволяє використання рефреш токену лише з того пристрою, для якого він був створений.

Токен доступу є JWT [31] токеном, підписаним за допомогою алгоритму SHA256 [7]. У тілі токена зберігається унікальний ідентифікатор користувача типу Guid, тривалість періоду його валідності в мілісекундах і дата його закінчення. Підписом кожного токена є конкатенація секретного ключа сервера, який однаковий для всіх і є незмінним, та рефреш токена.

Відповідно лише користувач, який залогінився на своєму пристрої зможе використовувати та оновлювати свої токени доступу.

Сам процес авторизації для Bearer схеми дуже схожий на процес для ApiKey, але з деякими відмінностями. Під час перевірки авторизаційного токена доступу, спочатку з нього дістається ідентифікатор користувача, а з кукі — ідентифікатор пристрою. За цими двома параметрами знаходиться користувач та відповідний вказаному пристрою рефереш токен.

Валідація самого токена доступу проводиться на основі перевірки його підпису з використанням, отриманих з бази даних ідентифікатора пристрою і рефреш токену, зконкатенованого із серверним ключем. Після перевірки підпису процес валідації продовжується, перевіряючи тривалість життя токену доступу, якщо вона вичерпалась, то такий токен вже не є валідним і запит закінчується з помилкою і статус кодом 401.

Варто оглянути розроблену блок-схему алгоритму авторизації в системі з додатку Д. Блоки 4 - 8 відносяться до авторизації за серверним ключем, а блоки 9 – 12 — за JWT токеном.

Блок 1 — отримання посилання на об’єкт запиту.

Блок 2 — отримання з об’єкту запиту авторизаційного хедера.

Блок 3 — перевірка чи схема хедера рівна схемі Bearer, якщо так, то перейди до блоку 9, інакше перейти до блоку 4.

Блок 4 — перевірка чи схема хедера рівна схемі ApiKey, якщо ні, то в системі не існує відповідного процесора схем, викидається виключення.

Блок 5 — отримання авторизаційних даних користувача за серверним ключем.

Блок 6 — перевірка чи користувача не знайдено, якщо так, то запит не є авторизованим, викидається виключення.

Блок 7 — перевірка чи ключ із запиту збігається із ключем користувача, якщо ні, то запит не є авторизованим, викидається виключення.

Блок 8 — перевірка чи IP адреса запиту входить у список дозволених, якщо ні, то запит не є авторизованим, викидається виключення, якщо так, то запит вважаться авторизованим.

Блок 9 — отримання авторизаційних даних користувача за ідентифікатором користувача із JWT токена.

Блок 10 — перевірка чи користувача не знайдено, якщо так, то запит не є авторизованим, викидається виключення.

Блок 11 — валідація JWT токена на основі отриманих авторизаційних даних користувача.

Блок 12 — якщо результат факт валідності токену підтверджено, то запит вважаться авторизованим, якщо ні, то запит не є авторизованим, викидається виключення.

Після успішної авторизації у системі, HTTP запит користувача перетворюється у відповідну йому команду або запит CQRS. Вони виконуються обробниками, що визначені в рамках API сервісу і взаємодіють з базою даних для запису або читання даних, після їх виконання публікуються події, обробники яких відправляють на виконання команди до інших сервісів у системі.

4.4 Сервіс планування задач

Сервіс використовує Quartz .Net для планування задач і базу даних PostgreSQL у якості сховища для задач та тригерів. Оскільки Quartz .Net не має вбудованого засобу створення бази даних та конфігурації її схеми, то для таких цілей, кожен екземпляр сервісу планування на старті перевіряє чи існує база даних для сховища задач. У випадку її відсутності він створює її, ініціалізує необхідні таблиці і тільки після цього запускає процес планування з увімкненим режимом кластеризації, маючи власний ідентифікатор, який встановлюється при розгортанні через змінні середовища.

Налаштування роботи Quartz .Net та обмежень CQRS для RabbitMQ винесені у файли JSON конфігурації та зображені на рисунку 4.6, де в якості сховища задач використовується провайдер для PostgreSQL, кожна таблиця в базі даних буде мати префікс qrtz, хоча і для роботи сховища виділено окрему базу даних, провайдером для серіалізації виступає Newtonsoft.JSON, ідентифікатор екземпляра в кластері визначається автоматично на основі змінних оточення, що передаються зовні, і стандартне обмеження на використання класу ThreadPool, з простору імен System.Threading, встановлене рівним 10.

```
"RabbitMqRpcConfiguration": {
  "CommandPrefetchCount": 10,
  "CommandConsumersCount": 10,
  "CommandExchangeName": "CommandExchange",
  "QueryPrefetchCount": 10,
  "QueryConsumersCount": 15,
  "QueryExchangeName": "QueryExchange",
  "EventPrefetchCount": 10,
  "EventConsumersCount": 15,
  "EventExchangeName": "EventExchange"
},
"QuartzConfiguration": {
  "ConfigMap": {
    "quartz.jobStore.type": "Quartz.Impl.AdoJobStore.JobStoreTX, Quartz",
    "quartz.jobStore.driverDelegateType": "Quartz.Impl.AdoJobStore.PostgreSQLDelegate, Quartz",
    "quartz.jobStore.tablePrefix": "qrtz_",
    "quartz.jobStore.lockHandler.type": "Quartz.Impl.AdoJobStore.SimpleSemaphore, Quartz",
    "quartz.jobStore.useProperties": "true",
    "quartz.jobStore.dataSource": "default",
    "quartz.dataSource.default.provider": "Npgsql",
    "quartz.serializer.type": "json",
    "quartz.scheduler.instanceId": "AUTO",
    "quartz.jobStore.clustered": "true"
  },
```

Рисунок 4.6 — Конфігурація сервісу планування

Для роботи шини команд встановлено обмеження в 10 паралельних отримувачів з черги, які можуть отримувати не більше 10 повідомлень наперед, що дозволить набагато швидше розвантажувати чергу RabbitMQ, та набагато більше і рівномірніше завантажити сервіс. Для шин запитів та подій ліміт отримання повідомлень один читачем рівний 15 оскільки такі повідомлення частіше будуть мати інформативний характер та будуть навантажувати систему менше ніж команди.

Для роботи з планувальником створено три команди, які дозволяють планувати виконання інших команд. Це забезпечується збереженням даних про задачу у вигляді серіалізованого Strongy-typed JSON.

ScheduleCommandExecutionCommand призначена для планування виконання нової команди після її валідації та збереження в API сервісі. У якості команди може бути запланована і виконана будь яка команда, що реалізує інтерфейс ICommand.

Кожна задача, запланована цією командою, додається в групу, де назвою є тип запланованої команди зконкатенований з ідентифікатором користувача, а ідентифікатором JobId типу Guid отриманий із команди.

Після успішного додавання в планувальник нової задачі, обробник публікує JobScheduledEvent, який повідомляє всіх про те, що задача успішно обробилась.

UnscheduleCommandExecutionCommand призначена для відміни виконання запланованої команди. У випадку, якщо задача на виконання не знайдена в планувальнику за вказаним ідентифікатором та групою, то обробник команди кидає виключення про неможливість знайти задачу, яку необхідно відмінити. У випадку успішної відміни задачі, обробник команди публікує JobUnscheduledEvent, який сигналізує про те, що всі оповіщення повинні бути позначені як відмінені.

UpdateScheduledCommandExecutionCommand призначена для оновлення даних по запланованій задачі. У випадку, якщо задача на

виконання не знайдена в планувальнику за вказаним ідентифікатором та групою, то обробник команди кидає виключення про неможливість знайти задачу, яку необхідно оновити. Сам обробник для оновлення даних по задачі, використовує повний перезапис цих даних, тобто фактично можливо за існуючим тригером, після перших кількох ітерацій виконання команди на відправку HTTP оповіщення, замінити його на Email оповіщення. Після успішного оновлення задачі обробник публікує ScheduledJobUpdatedEvent, який дозволяє оновити інформацію про цю задачу в усіх модулях системи.

Для пришвидшення роботи з даними запланованих задач, усі задачі не видаляються з сховища в PostgreSQL, що дозволяє зекономити час на видалення та ресурси центрального процесора на перерахунок індексів на таблицях з задачами та тригерами.

Виконання запланованої задачі супроводжується публікацією трьох можливих подій:

- JobExecutionStarted, завжди публікується зразу після старту виконання задачі, API сервіс на обробник цієї події створює новий запис в таблиці job_executions із статусом Started;
- JobExecutionSucceed публікується у випадку успішного виконання будь-якої задачі, під успішним мається на увазі відсутність викинутого виключення у процесі виконання, обробник події на строні API сервісу, оновлює стан в таблиці job_executions до статусу Succeed;
- JobExecutionFailed публікується у випадку викидання в процесі виконання виключення, яке перериває виконання задачі, обробник події в API сервісі, відзначає це зі статусом Failed;

4.5 Сервіс відправки Email оповіщень

Даний сервіс використовує має лише одну команду SendEmailNotificationCommand, за допомогою якої сервіс планування виконує задачу на відправку електронного листа.

Для відправки електронних листів використовується клас SmtpClient з простору імен System.Net.Mail бібліотек CoreFX. Для створення SmtpClient було створено фабрику, яка зареєстрована в контейнері Autofac як SingleInstance, що дозволяє забезпечити ініціалізацію її об’єкта лише одного разу. Дана фабрика створює клієнт для роботи по SMTP протоколу в залежності від переданих параметрів. У випадку, якщо в параметрах на створення клієнту не вказаний SMTP сервер, то буде використано стандартний сервер системи, доступ до якого конфігурується через JSON файл.

Команда відправки електронного листа не може містити шаблон повідомлення та усіх відправників, тому що вона зберігається в сховищі у сервісі планування задач, який і так використовує багато ресурсів центрального процесора та пам’яті для забезпечення процесу точного планування, а якщо додати іще і процес збереження шаблонів розміром у кілька тисяч кілобайт, то методи горизонтального масштабування можуть виявитись неефективними в майбутньому, тому команда містить лише ідентифікатори необхідний рядків у таблицях бази даних API, а під час обробки команди на відправку електронного листа, обробник спочатку відправить запит EmailNotificationDetailsQuery, який поверне йому необхідні для відправки деталі.

Після отримання деталей оповіщення, для всіх отримувачів повідомлення компілюється HTML шаблон, в який підставляються дані отримувача в якості змінних, що дозволяє створювати відправляти персоналізовані електронні листи.

Обробник команди в результаті відправки Email оповіщення публікує дві події:

– EmailNotificationSendingFailedEvent, який сигналізує про те, що в процесі з'єднання та взаємодії із SMTP сервером відбулась помилка, яка призвела до переривання процесу відправки;
– EmailNotificationSentEvent, який сигналізує про успішне завершення процесу відправки Email оповіщення;

4.6 Сервіс виконання HTTP запитів

Оскільки даний сервіс призначений виключно для відправки HTTP оповіщень, то взаємодія з ним відбувається за допомогою єдиної команди SendHttpNotificationCommand, яка викликається із сервісу планування.

Для відправки HTTP запитів використовується клас HttpClient із простору імен System.Net.Http з бібліотек CoreFX. Для роботи з ним було створено фабрику, яка відповідальна за створення об'єктів HttpClient з використанням додаткових хедерів HTTP запиту. Варто зауважити, що об'єкт фабрики ініціалізується лише одного разу на старті застонку, для чого використовується тип реєстрації SingleInstance в контейнері Autofac.

На початку обробки команди, обробник формує посилання із трьох параметрів: схеми запиту, доменного імені сервера та шляху в контексті його доменного імені. У випадку, якщо запит містить query параметри, то за допомого методу Aggregate класу StringBuilder формується список параметрів вигляду: key1=value1&key2=value2.

Для всіх параметрів у посиланні використовується кодування URL кодування, яке замінює усі невалідні для посилання символи на їх аналоги, що дозволяє відправляти в параметрах посилання будь-який набір тексту без можливих помилок.

Обробник команди SendHttpNotificationCommand використовує, згадану вище, фабрику для кожного HTTP оповіщення, яке необхідно відправити, передаючи в фабрику список HTTP хедерів оповіщення, якщо вони доступні для відправки.

У результаті відправки HTTP оповіщення може бути опубліковано дві події:

- HttpNotificationSentEvent, сигналізує про успішну відправку оповіщення, навіть якщо статус код отриманий у відповіді не є успішним, тобто не входить в інтервал від 200 до 299;
- HttpNotificationSendingFailedEvent, сигналізує про помилку в процесі відправки оповіщення, якою найчастіше може бути недоступність сервера, на який відправляється оповіщення або ж невалідне DNS ім’я, яке було змінене у той момент, коли оповіщення вже було заплановане;

Висновки до розділу 4

У даному розділі було викладено процес розробки системи віддаленого планування оповіщень, починаючи з розробки структури проекту та його архітектури, закінчуючи реалізацією конкретних функціональних можливостей.

Кожен підрозділ даного розділу розглядає основні проблеми, які були вирішені під час розробки та описує розроблені компоненти системи, які під час композиції складають цілісний застосунок.

Використання принципу CQRS для організації взаємодії між компонентами системи, виявилось надзвичайно зручним. Розділення відповідальності між командами та запитами надало програмному коду системи чіткої структурованості. Використання CQRS також забезпечило

принцип прозорості розміщення, який виявився дуже зручним, адже виклик віддаленої команди нічим не відрізняється від виклику локальної.

Результат розробки можна вважати успішним, адже на виході отримано систему, що відповідає технічним та функціональним вимогам викладеним у розділі 2.

Відповідно до вимог з розділу 2, система має автоматично згенеровану онлайн документацію, що відображає публічний API, який зображено на рисунку 4.7.

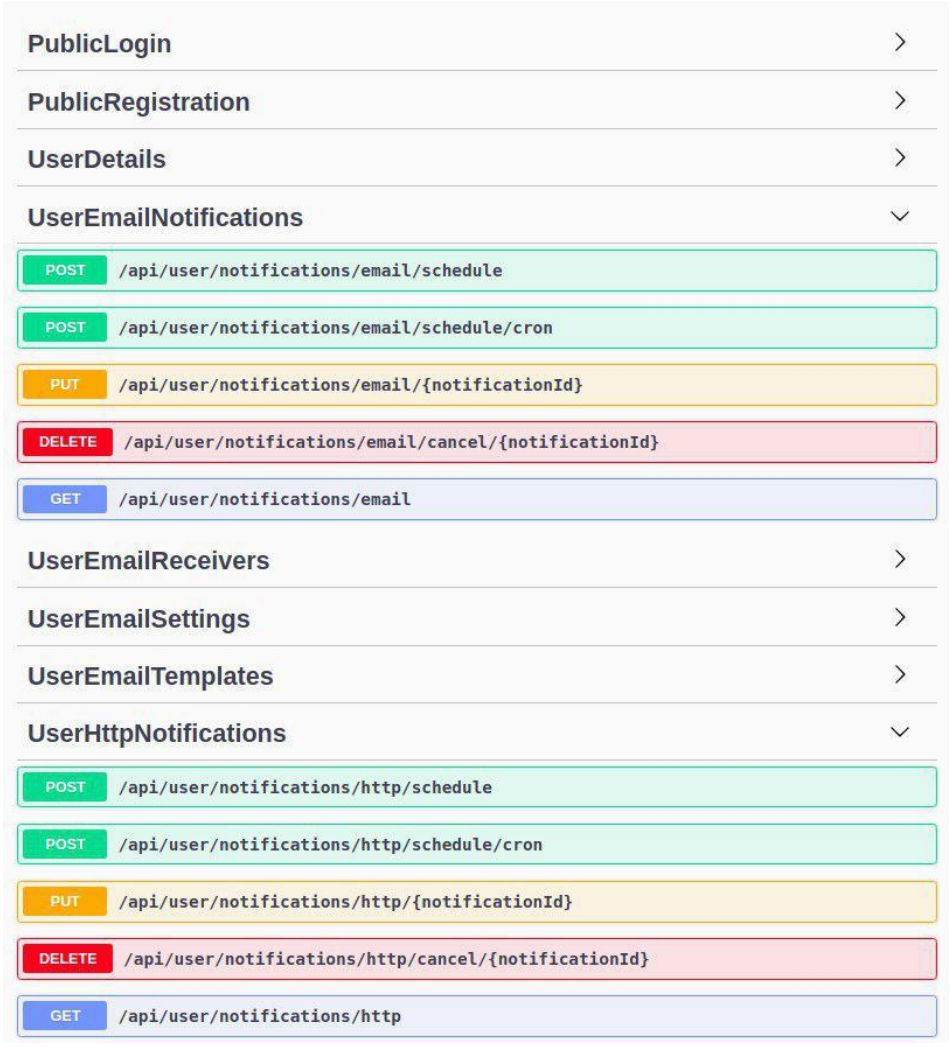

Рисунок 4.7 — Онлайн документація публічного API системи віддаленого планування оповіщень

5. ТЕСТУВАННЯ СИСТЕМИ

Для перевірки працездатності системи необхідно провести три тести:

– надсилання HTTP оповіщення щохвилини;

– надсилання Email оповіщення щохвилини;

– навантажувальний тест на отримання списку HTTP оповіщень;

Варто зазначити, що при тестуванні відправки оповіщень з певним інтервалом потрібно ввести допустиму похибку в 1 секунду, адже досить багато часу може бути витрачено на передачу оповіщень через мережу інтернет.

Для перевірки модуля HTTP оповіщень було використано ресурс webhook.site, який дозволяє відправляти на нього оповіщення і переглядати їх контент.

Тестування варто почати з HTTP оповіщень. Як можна бачити з рисунку 5.1, було створено оповіщення, яке має HTTP метод POST, на домен webhook.site з тестовими даними для відправки.

Рисунок 5.1 — Створення HTTP оповіщення

Оповіщення повинно відправитись після вказаного в полі startAt часу з інтервалом у 15 секунд. Також у ньому присутні хедери та query параметри, які теж повинні прийти у повідомленні на сайті webhook.site.

На рисунку 5.2 зображено список запитів, які прийшли на сервіс webhook.site, як можна бачити усі дані оповіщення правильні, а інтервал між доставкою відповідає встановленому в рамках похибки.

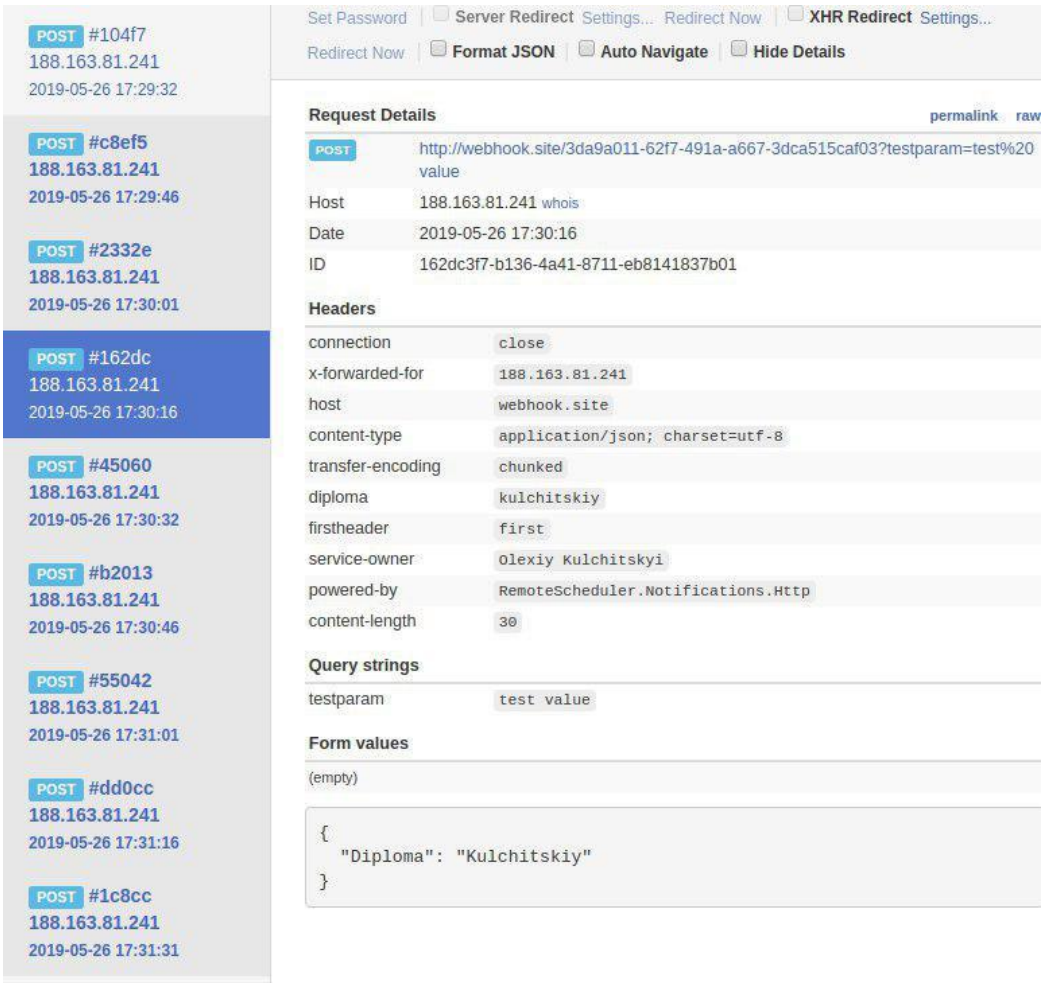

Рисунок 5.2 — Отримані HTTP оповіщення

Для перевірки модуля відправки Email оповіщень було використано ресурс tempail.com, який дозволяє скористатись тимчасовою електронною поштою, на яку будуть надсилатись оповіщення. На рисунку 5.3 зображено створення Email оповіщення. Як можна бачити оповіщення буде відправлене з темою «test diploma message», 10 разів з інтервалом у 5 секунд.

Рисунок 5.3 — Створення Email оповіщення

Варто зауважити, що даний сервіс не дає можливості відсортувати за часом надходження повідомлення, час який зображений у таблиці показує різницю між часом надходження та теперішнім. Попри це, з рисунку 5.4 видно, що інтервал між надходженням повідомлень складав 5 секунд з урахуванням допустимої похибки.

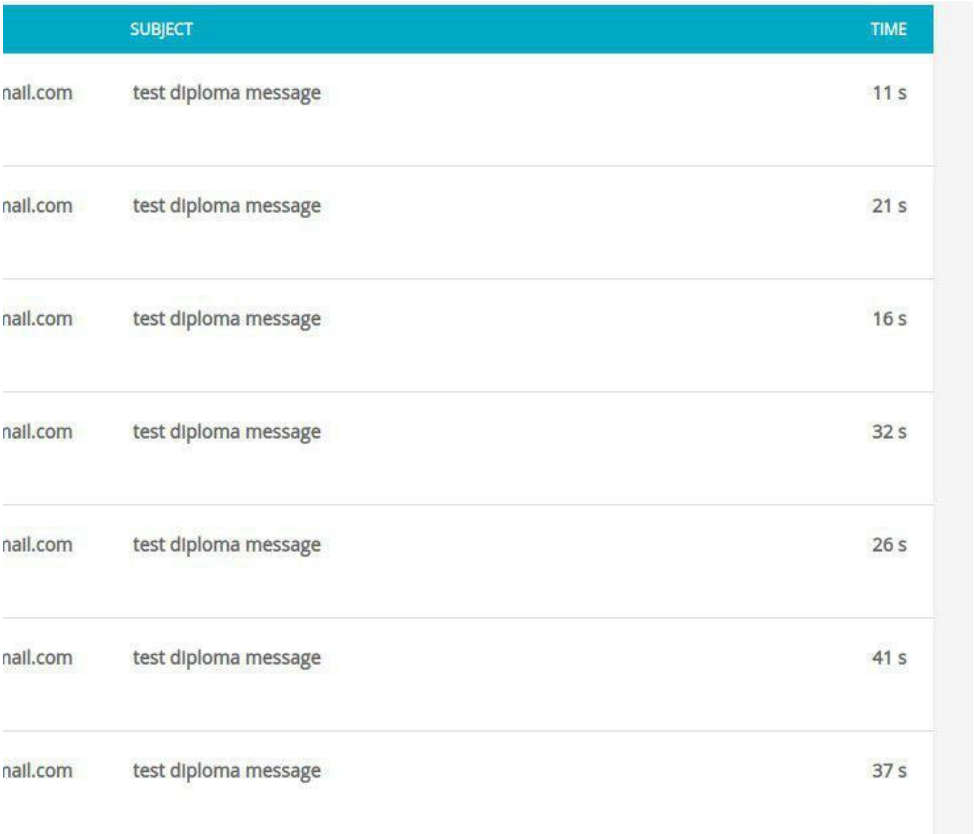

Рисунок 5.4 — Список отриманих оповіщень сервісу tempail.com

Для навантажувального тестування системи скористаємось утилітою Apache JMeter версії 5.1.1, вона дозволяє досить гнучко налаштувати запит та виконує підрахунок багатьох параметрів. У цьому тесті ціллю є 100 запитів на секунду, адже саме ця цифра прописана у вихідних даних до проекту.

Тестування буде проводитись для запиту /api/user/notifications/http, який повертає список HTTP оповіщень, з приблизним розміром відповіді 10 кілобайт, у кількості 10000 запитів з двох паралельних клієнтів. Характеристики машини, на якій буде проведено тестування:

- процесор Intel Core i5 7200, 2 ядра, 4 потоки;
- 8 Гб, ОЗУ DDR4,
- 256 Гб SSD

Як можна бачити з рисунку 5.5 для 10190 запитів, середня пропускна здатність була рівною 147.5 запитів на секунду, при похибці в 3.83 запитів на секунду. Такі цифри є значно більшими за 100 запитів на секунду та задовольняють вихідні вимоги до проекту.

# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
10190	13	5	72	3.83	0.02%	147.5/sec	1499.38	72.72	10410.5
10190	13	5	72	3.83	0.02%	147.5/sec	1499.38	72.72	10410.5

Рисунок 5.5 — Таблиця результатів тестування

На рисунку 5.6 зображено графік розподілу часу відповіді. Більшість запитів отримали відповідь в проміжку від 0 до 20 мілісекунд, у той час як більше ніж 300 запитів отримували відповідь від 20 до 80 мілісекунд. Така розбіжність може бути пов’язана із процесом збирання сміття в самому додатку, адже дуже велика кількість запитів до системи, генерує величезну кількість серіалізованих даних, які збирач сміття повинен видалити. Але така розбіжність не є критичною та дорівнює середній затримці при передачі даних через мережу.

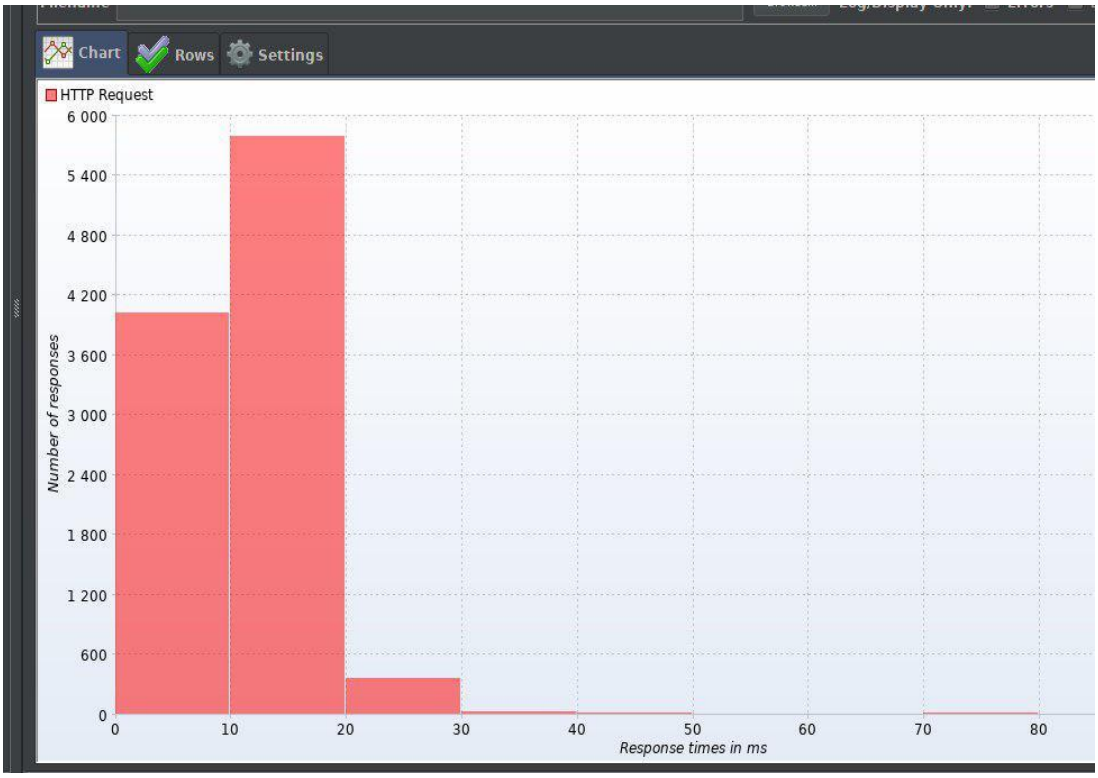

Рисунок 5.6 — Графік розподілу часу відповіді

Висновки до розділу 5

У даному розділі було розглянуто процес тестування розробленої системи віддаленого планування оповіщень.

Для тестування були обрані головні модулі системи, які повинні працювати точно та стабільно для забезпечення високої кількості одночасно запланованих оповіщень. Робота сервісу планування показала свою стабільність та забезпечила надсилання оповіщень за встановленим графіком: кожні 15 секунд для HTTP оповіщень і кожні 5 секунд для Email оповіщень.

Також система успішно пройшла навантажувальний тест, показавши результат більшій ніж було очікувано у вихідних даних до проекту. Варто зауважити, що тестування відбувалось на робочій машині, а в умовах реального серверу результати будуть значно кращими.

ВИСНОВКИ

Результатом бакалаврського проекту стала запрограмована система віддаленого планування оповіщень, що відповідає усім висунутим технічних та функціональним вимогам. Система була успішно протестована, а навантажувальний тест показав, що її можливості перевищують заявлені у вихідних даних.

Реалізовані функціональні можливості в парі з відсутністю ціни за використання, роблять розроблену систему конкурентоспроможною відносно існуючих рішень.

У ході виконання бакалаврського проекту було проаналізовано сучасні методи планування оповіщень, в рамках аналізу визначено основні типи оповіщень та оглянуто системи аналоги. На основі функціональних можливостей систем аналогів було сформовано перелік технічних та функціональних вимог до проекту.

На основі висунутих вимог було обрано технології, за допомогою яких буде реалізовано проект та оглянуто принципи проектування сучасних застосунків, які необхідно використати в проекті.

За допомогою сучасних шаблонів проектування була розроблена архітектура системи, яка є пристосованою до можливих змін, що дозволяє істотно спростити процес додавання нових типів оповіщень. У випадку необхідності масштабування системи мікросервісна архітектура значно спрощує цей процес і для того щоб розгорнути один або кілька сервісів системи на іншій машині потрібно лише кілька хвилин.

Використана платформа .NET Core дозволяє розгортати систему на будь-якій сучасній серверній операційній системі, що може значно здешевити процес підтримки.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Академічний тлумачний словник української мови [Електронний ресурс]. — режим доступу: http://sum.in.ua/s/spovishhennja.
2. RFC 2616 - Hypertext Transfer Protocol HTTP/1.1 [Електронний ресурс]. — режим доступу: https://tools.ietf.org/html/rfc2616.
3. RFC 6749 - The OAuth 2.0 Authorization Framework [Електронний ресурс]. — режим доступу: https://tools.ietf.org/html/rfc6749.
4. RFC 5321 - Simple Mail Transfer Protocol [Електронний ресурс]. — режим доступу: https://tools.ietf.org/html/rfc5321.
5. Облачные продукты AWS [Електронний ресурс]. — режим доступу: https://aws.amazon.com/products.
6. AWS Lambda – Бессерверные вычисления – Amazon Web Services [Елктронний ресурс]. — режим доступу: https://aws.amazon.com/lambda.
7. Hashing with SHA-256 [Елетронний ресурс]. — режим доступу: https://medium.com/biffures/part-5-hashing-with-sha-256-4c2afc191c40.
8. What is Horizontal Scaling? [Електронний ресурс]. — режим доступу: https://www.techopedia.com/definition/7594/horizontal-scaling.
9. RESTful API Authentication Basics [Електронний ресурс]. — режим доступу: https://blog.restcase.com/restful-api-authentication-basics/.
10. Cron Expressions [Електронний ресурс]. — режим доступу: https://docs.oracle.com/cd/E12058_01/doc/doc.1014/e12030/cron_expressions.htm.
11. .Net Microservices: Architecture for Containerized .Net Applications / César de la Torre, Bill Wagner, Mike Rousos — M: Microsoft Press, 2018 — 289c.
12. Микросервисы (Microservices) / Хабр [Електронний ресурс]. — режим доступу: https://habr.com/ru/post/249183/.

13. Monolithic Architecture pattern [Електронний ресурс]. — режим доступу: https://microservices.io/patterns/monolithic.html.
14. Using Docker: Developing and Deploying Software with Containers 1st Edition / Adrian Mouat — O'Reilly Media, 2016 — 354с.
15. Какая система виртуализации лучше? / Хабр [Електронний ресурс]. — режим доступу: https://habr.com/ru/post/70608/.
16. RabbitMQ in Depth 1st Edition / Gavin M. Roy — Manning Publications, 2017 — 264с.
17. Client Documentation — RabbitMQ [Електронний ресурс]. — режим доступу: https://www.rabbitmq.com/documentation.html.
18. RabbitMQ Hits One Million Messages Per Second [Електронний ресурс]. — режим доступу: https://content.pivotal.io/blog/rabbitmq-hits-one-million-messages-per-second-on-google-compute-engine.
19. CQRS [Електронний ресурс]. — режим доступу: https://martinfowler.com/bliki/CQRS.html.
20. DBMS (Database Management System) — Национальная библиотека им. Н. Э. Баумана [Електронний ресурс]. — режим доступу: https://ru.bmstu.wiki/DBMS_(Database_Management_System).
21. SQL Cookbook / Anthony Molinaro — O'Reilly Media, 2005 — 634с.
22. PostgreSQL — Национальная библиотека им. Н. Э. Баумана [Електронний ресурс]. — режим доступу: https://ru.bmstu.wiki/PostgreSQL
23. PostgreSQL: Documentation: 9.5: Index Types [Електронний ресурс]. — режим доступу: https://www.postgresql.org/docs/9.5/indexes-types.html
24. Принципы SOLID [Електронний ресурс]. – Режим доступу: https://metanit.com/sharp/patterns/5.1.php
25. Richter J. CLR via C# (4th Edition) (Developer Reference) / J. Richter – M: Microsoft Press, 2012 — 896с.

26. C# 7.0. Справочник. Полное описание языка, 7-е издание / Джозеф Албахари, Бен Албахари — Диалектика, 2016 — 1088с.
27. Strong and weak typing – Wikipedia [Електронний ресурс].— режим доступу: https://en.wikipedia.org/wiki/Strong_and_weak_typing
28. Dapper vs Entity Framework vs ADO.NET Performance Benchmarking [Електронний ресурс]. — режим доступу: https://exceptionnotfound.net/dapper-vs-entity-framework-vs-ado-net-performance-benchmarking/
29. Quartz.NET Features | Quartz.NET Documentation [Електронний ресурс]. — режим доступу: https://www.quartz-scheduler.net/features.html
30. OAI/OpenAPI-Specification [Електронний ресурс]. — режим доступу: http://github.com/OAI/OpenAPI-Specification/ blob/master/versions/ 3.0.0. md
31. RFC 7519 - JSON Web Token (JWT) [Електронний ресурс]. — режим доступу: https://tools.ietf.org/html/rfc7519

ДОДАТОК А

Структурна схема системи планування оповіщень. Обов’язковий.

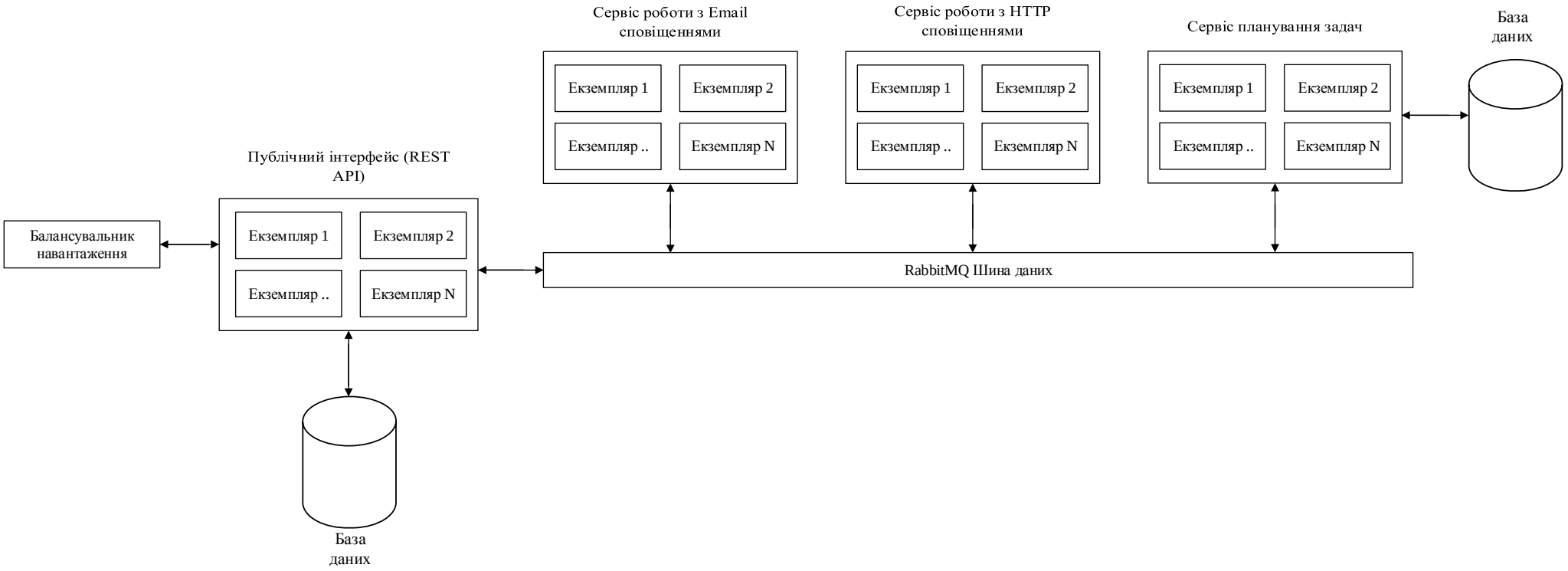

Рисунок А.1 — Структурна схема системи планування оповіщень

ДОДАТОК Б

Блок-схема загального алгоритму роботи системи. Обов’язковий.

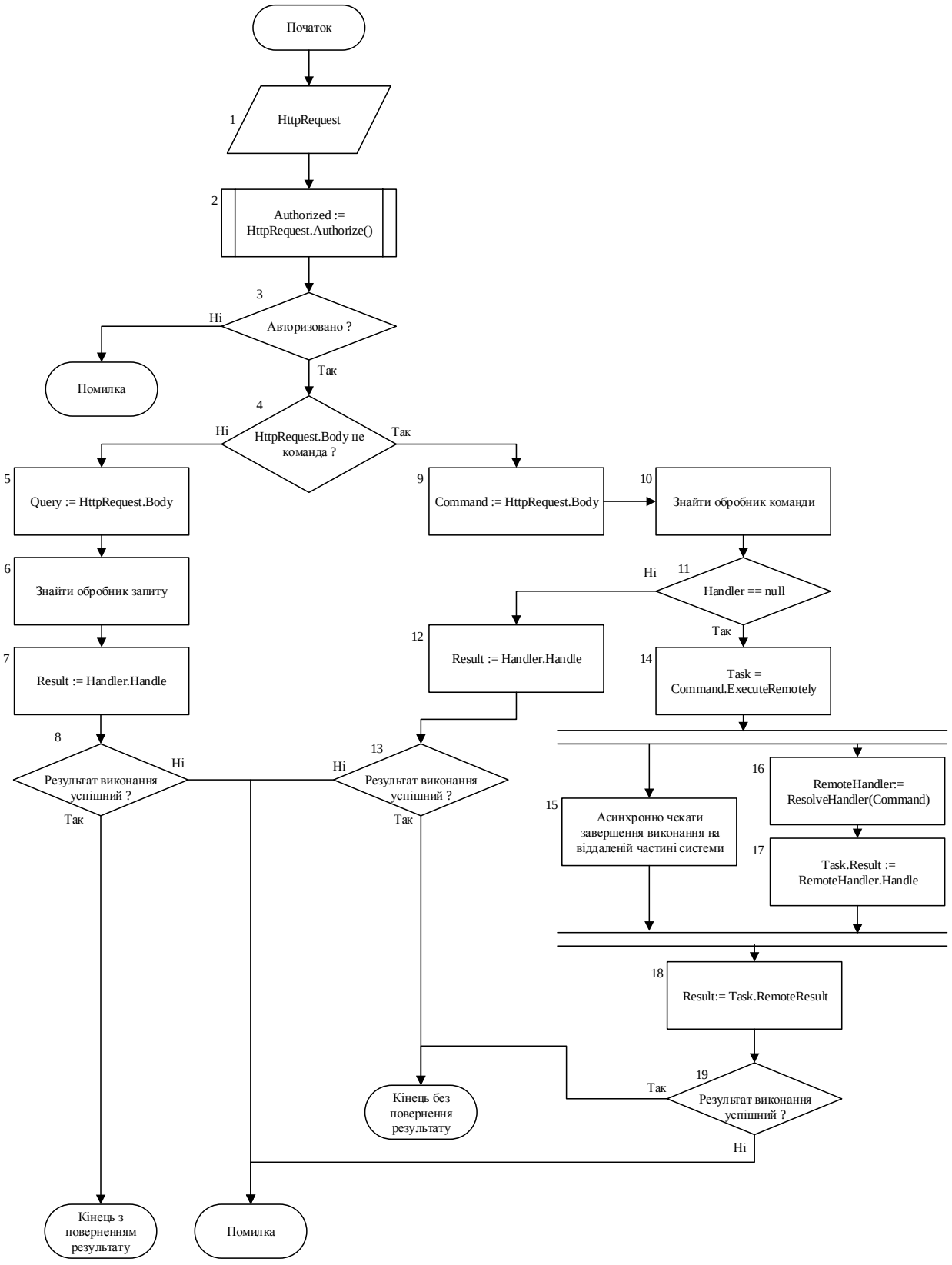

Рисунок Б.1 — Блок-схема загального алгоритму роботи системи

ДОДАТОК В

UML діаграма послідовності роботи принципу CQRS. Обов’язковий.

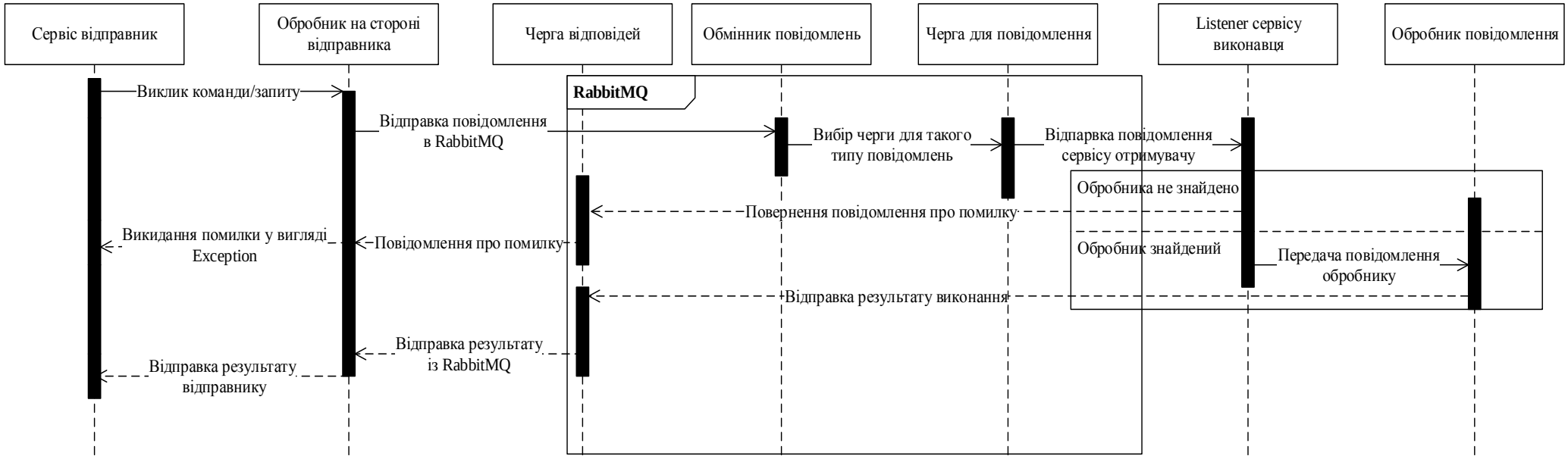

Рисунок В.1 — UML діаграма послідовності роботи принципу CQRS

ДОДАТОК Г

Структурна схема бази даних системи. Обов’язковий.

Рисунок Г.1 — Структурна схема бази даних системи

					ІТ51.100БАК.006 Д4
Ізм.	Лист	№ докум.	Підпис	Дата	

ДОДАТОК Д

Блок-схема алгоритму авторизації. Обов’язковий.

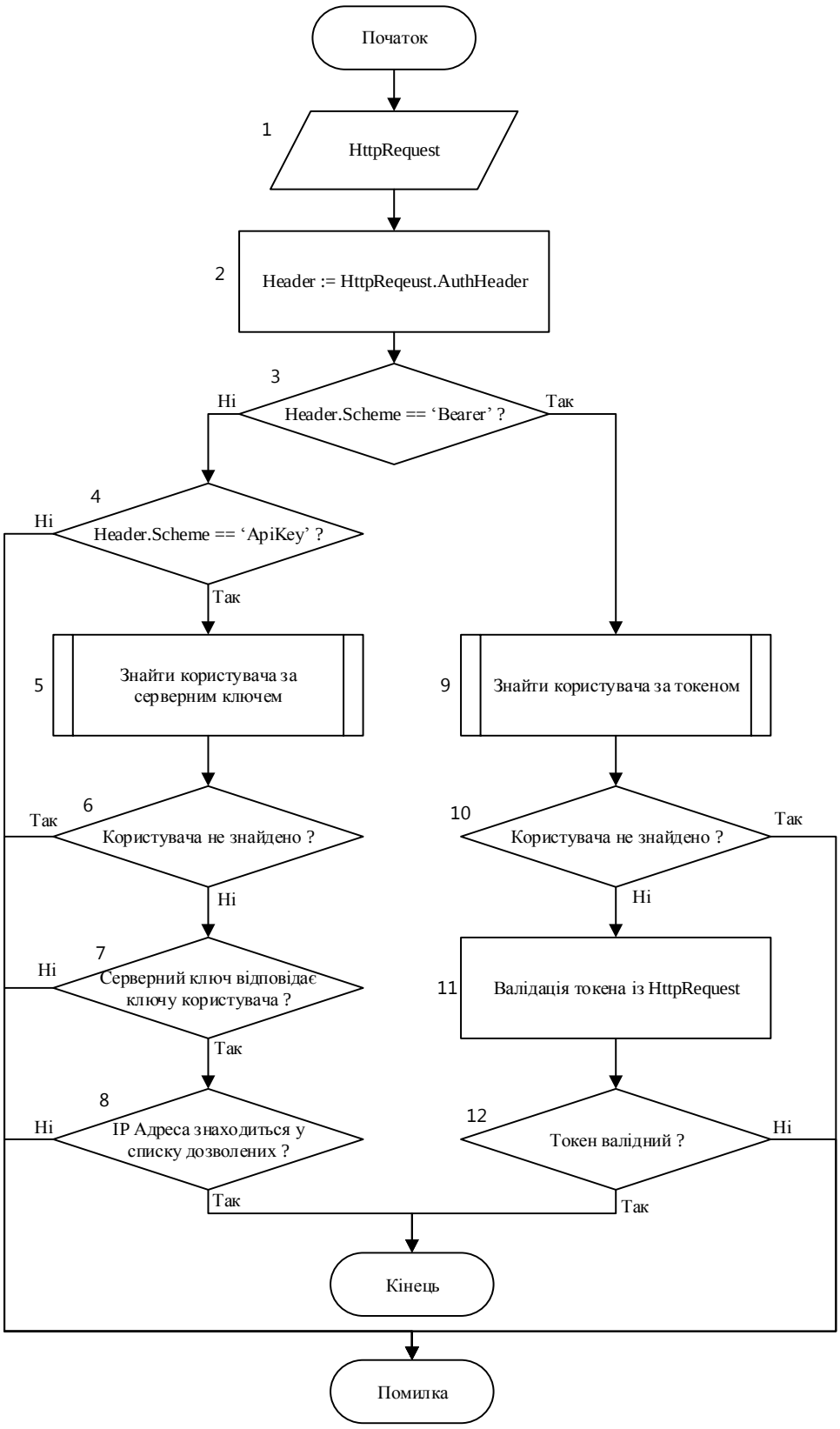

Рисунок Д.1 — Блок-схема алгоритму авторизації

ДОДАТОК Е

Вихідний код програми. Рекомендований.

```
using System;
namespace RemoteScheduler.Core.Configuration
{
   public sealed class Environment : IEnvironment
   {
      public string InstanceId => GetEnvironmentVariableValue("INSTANCE_ID");
      public string EnvironmentName => GetEnvironmentVariableValue("HOSTING_ENVIRONMENT");
      public string GetEnvironmentVariableValue(string variableName)
      {
         return System.Environment.GetEnvironmentVariable(variableName);
      }
      public bool IsDevelopment()
      {
         return string.Equals("Development", EnvironmentName, StringComparison.InvariantCultureIgnoreCase);
      }
      public bool IsStaging()
      {
         return string.Equals("Staging", EnvironmentName, StringComparison.InvariantCultureIgnoreCase);
      }
      public bool IsProduction()
      {
         return string.Equals("Production", EnvironmentName, StringComparison.InvariantCultureIgnoreCase);
      }
   }
}
using Autofac;
namespace RemoteScheduler.Core.Configuration
{
   public class EnvironmentModule : Module
   {
      protected override void Load(ContainerBuilder builder)
      {
         builder.RegisterType<Environment>()
            .AsImplementedInterfaces()
            .SingleInstance();
      }
   }
}
using Microsoft.Extensions.Configuration;
namespace RemoteScheduler.Core.Configuration
{
   public interface IConfiguration
   {
      IConfigurationSection ConfigurationSection { get; set; }
   }
}
namespace RemoteScheduler.Core.Configuration
{
   public interface IEnvironment
   {
      string InstanceId { get; }
      string EnvironmentName { get; }
      string GetEnvironmentVariableValue(string variableName);
      bool IsDevelopment();
      bool IsStaging();
      bool IsProduction();
   }
}
using System.Reflection;
namespace RemoteScheduler.Core.Configuration.Extensions
{
   public static class AssemblyExtensions
   {
      public static string AssemblyNamespace(this Assembly assembly)
      {
         return assembly.FullName.Split(',')[0];
      }
   }
}
using System;
namespace RemoteScheduler.Core.Cqrs.Abstractions
{
   public class MessageExecutionContext
   {
      public Predicate<Type> ExecutionFilter { get; set; }
   }
}
using System;
using System.Threading.Tasks;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace RemoteScheduler.Core.Cqrs.Abstractions.Buses
{
   public interface ICommandBus
   {
      Task Execute(ICommand command, MessageExecutionContext executionContext = null);
      Task Execute<TCommand>(TCommand command, MessageExecutionContext executionContext = null) where TCommand : ICommand;
   }
}
using System.Threading.Tasks;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace RemoteScheduler.Core.Cqrs.Abstractions.Buses
{
   public interface IEventBus
   {
      Task Raise(IEvent @event, MessageExecutionContext executionContext = null);
      Task Raise<TEvent>(TEvent @event, MessageExecutionContext executionContext = null) where TEvent : IEvent;
   }
}
using System.Threading.Tasks;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace RemoteScheduler.Core.Cqrs.Abstractions.Buses
{
   public interface IQueryBus
   {
      Task<IQueryResponse> Execute(IQuery query, MessageExecutionContext executionContext = null);
      Task<TResponse> Execute<TQuery, TResponse>(TQuery query, MessageExecutionContext executionContext = null)
         where TQuery : IQuery<TResponse>
         where TResponse : IQueryResponse;
   }
}
namespace RemoteScheduler.Core.Cqrs.Abstractions.Contracts
{
   public interface ICommand
   {
   }
}
namespace RemoteScheduler.Core.Cqrs.Abstractions.Contracts
{
   public interface IEvent
   {
   }
}
namespace RemoteScheduler.Core.Cqrs.Abstractions.Contracts
{
   public interface IQuery
```

```
    {
    }
    public interface IQuery<TResult> : IQuery where TResult :
IQueryResponse
    {
    }
}
namespace RemoteScheduler.Core.Cqrs.Abstractions.Contracts
{
    public interface IQueryResponse
    {
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
using RemoteScheduler.Core.Exceptions;
namespace RemoteScheduler.Core.Cqrs.Abstractions.Exceptions
{
    public abstract class CqrsException :
RemoteSchedulerException/
    {
    }
    public class InvalidCommandHandlerException : CqrsException
    {
        public InvalidCommandHandlerException(ICommand
command, Type handlerType)
        {
            Command = command;
            HandlerType = handlerType;
        }
        public ICommand Command { get; }
        public Type HandlerType { get; }
    }
    public class InvalidQueryHandlerException : CqrsException
    {
        public InvalidQueryHandlerException(IQuery query, Type
handlerType)
        {
            Query = query;
            HandlerType = handlerType;
        }
        public IQuery Query { get; }
        public Type HandlerType { get; }
    }
}
using System.Threading.Tasks;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Commands
{
    public abstract class CommandHandlerBase<TCommand> :
ICommandHandler<TCommand> where TCommand : ICommand
    {
        public abstract Task Handle(TCommand command);
        public Task Handle(ICommand command)
        {
            return Handle((TCommand) command);
        }
    }
}
using System.Threading.Tasks;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Commands
{
    public interface ICommandHandler
    {
        Task Handle(ICommand command);
    }
    public interface ICommandHandler<in TCommand>:
ICommandHandler where TCommand : ICommand
    {
        Task Handle(TCommand command);
    }
}
using System.Threading.Tasks;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Events
{
    public abstract class EventSubscriberBase<TEvent> :
IEventSubscriber<TEvent> where TEvent : IEvent
    {
        public abstract Task Handle(TEvent @event);
        public Task Handle(IEvent @event)
        {
            return Handle((TEvent) @event);
        }
    }
}
using System.Threading.Tasks;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Events
{
    public interface IEventSubscriber
    {
        Task Handle(IEvent @event);
    }
    public interface IEventSubscriber<in TEvent> : IEventSubscriber
where TEvent : IEvent
    {
        Task Handle(TEvent @event);
    }
}
using System.Threading.Tasks;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Queries
{
    public interface IQueryHandler
    {
        Task<IQueryResponse> Handle(IQuery query);
    }
    public interface IQueryHandler<in TQuery, TResult> :
IQueryHandler
        where TQuery : IQuery<TResult>
        where TResult : IQueryResponse
    {
        Task<TResult> Handle(TQuery query);
    }
}
using System.Threading.Tasks;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Queries
{
    public abstract class QueryHandlerBase<TQuery, TResult> :
IQueryHandler<TQuery, TResult>
        where TResult : IQueryResponse where TQuery :
IQuery<TResult>
    {
        public abstract Task<TResult> Handle(TQuery query);
        public async Task<IQueryResponse> Handle(IQuery query)
        {
            return await Handle((TQuery) query);
        }
    }
}
using Autofac;
using RemoteScheduler.Core.Cqrs.Implementation.Buses;
namespace RemoteScheduler.Core.Cqrs.Implementation
{
    public class CqrsModule : Module
    {
        protected override void Load(ContainerBuilder builder)
        {

builder.RegisterType<CommandBus>().AsImplementedInterfaces()
;

builder.RegisterType<QueryBus>().AsImplementedInterfaces();
            builder.RegisterType<EventBus>()
                .AsImplementedInterfaces()
```

```
            .SingleInstance();
        }
    }
}
using System.Threading.Tasks;
using Autofac;
using RemoteScheduler.Core.Cqrs.Abstractions;
using
RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Commands;
namespace RemoteScheduler.Core.Cqrs.Implementation.Buses
{
    public class CommandBus : ICommandBus
    {
        private readonly ILifetimeScope _lifetimeScope;
        public CommandBus(ILifetimeScope lifetimeScope)
        {
            _lifetimeScope = lifetimeScope;
        }
        public Task Execute(ICommand command,
MessageExecutionContext executionContext = null)
        {
            var handlerType =
typeof(ICommandHandler<>).MakeGenericType(command.GetTy
pe());
            var commandHandler = (ICommandHandler)
_lifetimeScope.Resolve(handlerType);
            if (executionContext != null)
            {
                if
(!executionContext.ExecutionFilter.Invoke(handlerType))
                {
                    throw new
InvalidCommandHandlerException(command, handlerType);
                }
            }
            return commandHandler.Handle(command);
        }
        public Task Execute<TCommand>(TCommand command,
MessageExecutionContext executionContext = null) where
TCommand : ICommand
        {
            var commandHandler =
_lifetimeScope.Resolve<ICommandHandler<TCommand>>();
            if (executionContext != null)
            {
                var handlerType = commandHandler.GetType();
                if
(!executionContext.ExecutionFilter.Invoke(handlerType))
                {
                    throw new
InvalidCommandHandlerException(command, handlerType);
                }
            }
            return commandHandler.Handle(command);
        }
    }
}
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Autofac;
namespace RemoteScheduler.Core.Cqrs.Implementation.Buses
{
    public class EventBus : IEventBus
    {
        private readonly ILifetimeScope _lifetimeScope;
        private readonly ILogger<EventBus> _logger;
        public EventBus(ILifetimeScope lifetimeScope,
ILogger<EventBus> logger)
        {
            _lifetimeScope = lifetimeScope;
            _logger = logger;
        }
        public Task Raise(IEvent @event, MessageExecutionContext
executionContext = null)
        {
            Task.Run(async () =>
            {
                using (var scope =
_lifetimeScope.BeginLifetimeScope("Event bus execution scope"))
                {
                    var subscriberType =
typeof(IEventSubscriber<>).MakeGenericType(@event.GetType())
;
                    var allSubscribersType =
typeof(IEnumerable<>).MakeGenericType(subscriberType);
                    var subscribers = (IEnumerable)
scope.Resolve(allSubscribersType);
                    var executionTasks = subscribers
                        .OfType<IEventSubscriber>()
                        .Where(subscriber =>
                        {
                            if (executionContext == null)
                            {
                                return true;
                            }
                            return
executionContext.ExecutionFilter.Invoke(subscriber.GetType());
                        })
                        .Select(subscriber =>
                            Task.Run(async () =>
                            {
                                try
                                {
                                    await subscriber.Handle(@event);
                                }
                                catch (Exception e)
                                {
                                    _logger.Fatal(JToken.FromObject(@event),
e);
                                }
                            }));
                    await Task.WhenAll(executionTasks);
                }
            }).ContinueWith(continuationAction: task =>
            {
                if (task.Exception != null)
                {
                    _logger.Fatal(JToken.FromObject(@event),
task.Exception);
                }
            });
            return Task.CompletedTask;
        }
        public Task Raise<TEvent>(TEvent @event,
MessageExecutionContext executionContext = null) where TEvent
: IEvent
        {
            Task.Run(async () =>
            {
                using (var scope =
_lifetimeScope.BeginLifetimeScope("Event bus execution scope"))
                {
                    var subscribers =
scope.Resolve<IEnumerable<IEventSubscriber<TEvent>>>();
                    var executionTasks = subscribers
                        .Where(subscriber =>
                        {
                            if (executionContext == null)
                            {
                                return true;
                            }
                            return
executionContext.ExecutionFilter.Invoke(subscriber.GetType());
                        })
                        .Select(subscriber =>
                            Task.Run(async () =>
                            {
                                try
                                {
                                    await subscriber.Handle(@event);
                                }
```

```
                        catch (Exception e)
                        {
                            _logger.Fatal(JObject.FromObject(@event),
e);
                        }
                    }));
                await Task.WhenAll(executionTasks);
            }
        }).ContinueWith(continuationAction: task =>
        {
            if (task.Exception != null)
            {
                _logger.Fatal(JObject.FromObject(@event),
task.Exception);
            }
        });
        return Task.CompletedTask;
    }
  }
}
using System.Linq;
using System.Threading.Tasks;
namespace RemoteScheduler.Core.Cqrs.Implementation.Buses
{
  public class QueryBus : IQueryBus
  {
    private readonly ILifetimeScope _lifetimeScope;
    public QueryBus(ILifetimeScope lifetimeScope)
    {
      _lifetimeScope = lifetimeScope;
    }
    public Task<IQueryResponse> Execute(IQuery query,
MessageExecutionContext executionContext = null)
    {
      var queryType = query.GetType();
      var genericInterface = queryType.GetInterfaces().First(type
=>
        type.IsGenericType &&
type.GetGenericTypeDefinition() == typeof(IQuery<>));
      var handlerType =
typeof(IQueryHandler<,>).MakeGenericType(queryType,
        genericInterface.GenericTypeArguments.First(type =>
type.IsAssignableTo<IQueryResponse>()));
      var queryHandler = (IQueryHandler)
_lifetimeScope.Resolve(handlerType);
      if (executionContext != null)
      {
        if
(!executionContext.ExecutionFilter.Invoke(handlerType))
        {
          throw new InvalidQueryHandlerException(query,
handlerType);
        }
      }
      return queryHandler.Handle(query);
    }
    public Task<TResponse> Execute<TQuery,
TResponse>(TQuery query, MessageExecutionContext
executionContext = null)
      where TQuery : IQuery<TResponse> where TResponse :
IQueryResponse
    {
      var queryHandler =
_lifetimeScope.Resolve<IQueryHandler<TQuery, TResponse>>();
      if (executionContext != null)
      {
        var handlerType = queryHandler.GetType();
        if
(!executionContext.ExecutionFilter.Invoke(handlerType))
        {
          throw new InvalidQueryHandlerException(query,
handlerType);
        }
      }
      return queryHandler.Handle(query);
    }
  }
}
using System.Reflection;
using RemoteScheduler.Core.Configuration.Extensions;
namespace RemoteScheduler.Core.Cqrs.RabbitMq
{
  public static class Constants
  {
    public static readonly string EntryAssemblyNamespace =
Assembly.GetEntryAssembly().AssemblyNamespace();
  }
}
using System.Reflection;
using Autofac;
namespace RemoteScheduler.Core.Cqrs.RabbitMq
{
  public class CqrsRmqModule : Module
  {
    protected override void Load(ContainerBuilder builder)
    {
      builder.RegisterType<JsonMessageSerializer>()
        .AsImplementedInterfaces()
        .SingleInstance();
      builder.RegisterType<RpcResponseListener>()
        .AsImplementedInterfaces()
        .SingleInstance()
        .WithAttributeFiltering()
        .AutoActivate();
      builder.RegisterGeneric(typeof(RpcCommandHandler<>))
        .As(typeof(ICommandHandler<>))
        .WithAttributeFiltering();
      builder.RegisterGeneric(typeof(RpcQueryHandler<,>))
        .As(typeof(IQueryHandler<,>))
        .WithAttributeFiltering();
      builder.RegisterGeneric(typeof(RpcEventSubscriber<>))
        .As(typeof(IEventSubscriber<>))
        .WithAttributeFiltering();
      builder.Register(context =>
          context.Resolve<IConnectionFactory>()

.CreateConnection($"{Assembly.GetEntryAssembly().AssemblyNa
mespace()}.{ConnectionTag.Publish}"))
        .Keyed<IConnection>(ConnectionTag.Publish)
        .As<IConnection>()
        .SingleInstance();
      builder.Register(context =>
          context.Resolve<IConnectionFactory>()

.CreateConnection($"{Assembly.GetEntryAssembly().AssemblyNa
mespace()}.{ConnectionTag.Consume}"))
        .Keyed<IConnection>(ConnectionTag.Consume)
        .As<IConnection>()
        .SingleInstance();
      var consumeTags = new[]
      {
        ChannelTag.RpcConsume,
        ChannelTag.EventConsume,
        ChannelTag.QueryConsume,
        ChannelTag.CommandConsume
      };
      var publishTags = new[]
      {
        ChannelTag.EventPublish,
        ChannelTag.QueuePublish,
        ChannelTag.CommandPublish,
        ChannelTag.QueryResponsePublish,
        ChannelTag.CommandResponsePublish
      };
      foreach (var consumeTag in consumeTags)
      {
        builder.Register(context =>
context.ResolveKeyed<IConnection>(ConnectionTag.Consume).Cr
eateModel())
          .Keyed<IModel>(consumeTag)
          .As<IModel>()
          .SingleInstance()
          .ExternallyOwned();
      }
```

```
        foreach (var publishTag in publishTags)
        {
            builder.Register(context => context.ResolveKeyed<IConnection>(ConnectionTag.Publish).CreateModel())
                .Keyed<IModel>(publishTag)
                .As<IModel>()
                .SingleInstance();
        }
        builder.RegisterAssemblyTypes(ThisAssembly)
            .AssignableTo<RpcListenerBase>()
            .AsSelf()
            .WithAttributeFiltering()
            .SingleInstance()
            .AutoActivate()
            .OnActivated(args => ((RpcListenerBase)args.Instance).SetupListener());
    }
  }
}
namespace RemoteScheduler.Core.Cqrs.RabbitMq.Client
{
  public class RpcCommandHandler<TCommand> : CommandHandlerBase<TCommand> where TCommand : ICommand
  {
    private readonly IModel _model;
    private readonly IMessageSerializer _serializer;
    private readonly IRpcResponseListener _responseListener;
    private readonly int _millisecondsTimeout = 60000;
    public RpcCommandHandler([KeyFilter(ChannelTag.CommandPublish)] IModel model,
        IMessageSerializer serializer, IRpcResponseListener responseListener)
    {
      _model = model;
      _serializer = serializer;
      _responseListener = responseListener;
    }
    public override async Task Handle(TCommand command)
    {
      var routingKey = typeof(TCommand).FullName;
      var correlationId = Guid.NewGuid();
      var basicProperties = _model.CreateBasicProperties();
      basicProperties.CorrelationId = correlationId.ToString();
      basicProperties.Expiration = _millisecondsTimeout.ToString(CultureInfo.InvariantCulture);
      basicProperties.ReplyTo = _responseListener.QueueName;
      var msgBody = _serializer.Serialize(command);
      try
      {
        var responseTask = _responseListener.Subscribe(correlationId);
        _model.BasicPublish("CommandExchange", routingKey, basicProperties, msgBody);
        var firstExecutedTask = await Task.WhenAny(responseTask, Task.Delay(_millisecondsTimeout));
        if (firstExecutedTask != responseTask)
        {
          throw new InvalidMessageException();
        }
        var result = responseTask.Result;
        var response = _serializer.Deserialize(result.Body);
        if (response is ExceptionData data)
        {
          throw new InvalidMessageException();
        }
      }
      finally
      {
        _responseListener.Unsubscribe(correlationId);
      }
    }
  }
}
using System.Collections.Generic;
using System.Reflection;
using System.Threading.Tasks;
namespace RemoteScheduler.Core.Cqrs.RabbitMq.Client
{
  public class RpcEventSubscriber<TEvent> : EventSubscriberBase<TEvent> where TEvent : IEvent
  {
    private readonly IModel _model;
    private readonly IMessageSerializer _serializer;
    public RpcEventSubscriber([KeyFilter(ChannelTag.EventPublish)] IModel model, IMessageSerializer serializer)
    {
      _model = model;
      _serializer = serializer;
    }
    public override Task Handle(TEvent @event)
    {
      var routingKey = @event.GetType().FullName;
      var body = _serializer.Serialize(@event);
      var properties = _model.CreateBasicProperties();
      properties.Headers = new Dictionary<string, object>()
      {
        ["Source"]=Constants.EntryAssemblyNamespace
      };
      _model.BasicPublish("EventExchange", routingKey, properties, body);
      return Task.CompletedTask;
    }
  }
}
using System;
using System.Threading.Tasks;
using Autofac.Features.AttributeFilters;
using RabbitMQ.Client;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace RemoteScheduler.Core.Cqrs.RabbitMq.Client
{
  public class RpcQueryHandler<TQuery, TResult> : QueryHandlerBase<TQuery, TResult>
    where TQuery : IQuery<TResult> where TResult : IQueryResponse
  {
    private readonly IModel _model;
    private readonly IMessageSerializer _serializer;
    private readonly IRpcResponseListener _responseListener;
    private readonly int _millisecondsTimeout = 60000;
    public RpcQueryHandler([KeyFilter(ChannelTag.QueuePublish)] IModel model,
        IMessageSerializer serializer, IRpcResponseListener responseListener)
    {
      _model = model;
      _serializer = serializer;
      _responseListener = responseListener;
    }
    public override async Task<TResult> Handle(TQuery command)
    {
      var routingKey = typeof(TQuery).FullName;
      var correlationId = Guid.NewGuid();
      var basicProperties = _model.CreateBasicProperties();
      basicProperties.CorrelationId = correlationId.ToString();
      basicProperties.Expiration = _millisecondsTimeout.ToString(CultureInfo.InvariantCulture);
      basicProperties.ReplyTo = _responseListener.QueueName;
      var msgBody = _serializer.Serialize(command);
      try
      {
        var responseTask = _responseListener.Subscribe(correlationId);
        _model.BasicPublish("QueryExchange", routingKey, basicProperties, msgBody);
```

```
            var firstExecutedTask = await
Task.WhenAny(responseTask,
Task.Delay(_millisecondsTimeout));
            if (firstExecutedTask != responseTask)
            {
               throw new InvalidMessageException();
            }
            var result = responseTask.Result;
            var response = _serializer.Deserialize(result.Body);
            if (response is ExceptionData data)
            {
               throw new InvalidMessageException();
            }
            return (TResult) response;
         }
         finally
         {
            _responseListener.Unsubscribe(correlationId);
         }
      }
   }
}
using System;
using System.Threading.Tasks;
using RabbitMQ.Client.Events;
namespace
RemoteScheduler.Core.Cqrs.RabbitMq.Client.ResponseListening
{
   public interface IRpcResponseListener
   {
      string QueueName { get; }
      Task<BasicDeliverEventArgs> Subscribe(Guid correlationId);
      void Unsubscribe(Guid correlationId);
   }
}
using System;
using System.Collections.Concurrent;
using RemoteScheduler.Core.Cqrs.RabbitMq.Enums;
namespace
RemoteScheduler.Core.Cqrs.RabbitMq.Client.ResponseListening
{
   internal sealed class RpcResponseListener :
IRpcResponseListener, IDisposable
   {
      private readonly IModel _channel;
      private readonly ICollection<string> _consumerTags = new
List<string>();
      private readonly ConcurrentDictionary<Guid,
TaskCompletionSource<BasicDeliverEventArgs>> _subscriptions
=
         new ConcurrentDictionary<Guid,
TaskCompletionSource<BasicDeliverEventArgs>>();
      public string QueueName { get; }
      public
RpcResponseListener([KeyFilter(ChannelTag.RpcConsume)]
IModel channel, IEnvironment environment)
      {
         _channel = channel;
         QueueName =
$"{Assembly.GetEntryAssembly().AssemblyNamespace()}.RpcRe
sponseQueue.{environment.InstanceId}";
         _channel.BasicQos(prefetchSize: 0, prefetchCount: 50,
global: false);
         _channel.QueueDeclare(QueueName, true, false, false);
         for (int i = 0; i < 50; i++)
         {
            var consumer = new
AsyncEventingBasicConsumer(_channel);
            consumer.Received += ConsumerOnReceived;
            var consumerTag = _channel.BasicConsume(consumer,
QueueName, true);
            _consumerTags.Add(consumerTag);
         }
      }
#pragma warning disable 1998
      private async Task ConsumerOnReceived(object sender,
BasicDeliverEventArgs @event)
#pragma warning restore 1998
      {
         if (!Guid.TryParse(@event.BasicProperties.CorrelationId,
out var correlationId))
         {
            return;
         }
         if (!_subscriptions.TryGetValue(correlationId, out var tcs))
         {
            return;
         }
#pragma warning disable 4014
         Task.Run(() => tcs.SetResult(@event));
#pragma warning restore 4014
      }
      public Task<BasicDeliverEventArgs> Subscribe(Guid
correlationId)
      {
         var tcs = new
TaskCompletionSource<BasicDeliverEventArgs>();
         _subscriptions[correlationId] = tcs;
         return tcs.Task;
      }
      public void Unsubscribe(Guid correlationId)
      {
         if (_subscriptions.TryGetValue(correlationId, out var tcs))
         {
            Task.Run(() => tcs.TrySetCanceled());
         }
      }
      public void Dispose()
      {
         if (_subscriptions.Any())
         {
            foreach (var taskCompletionSource in _subscriptions)
            {
               Task.Run(() =>
taskCompletionSource.Value.TrySetCanceled());
            }
            _subscriptions.Clear();
         }
         foreach (var consumerTag in _consumerTags)
         {
            _channel?.BasicCancel(consumerTag);
         }
         _channel?.Dispose();
      }
   }
}
namespace RemoteScheduler.Core.Cqrs.RabbitMq.Configurations
{
   public interface IRabbitMqRpcConfiguration
   {
      ushort CommandPrefetchCount { get; }
      int CommandConsumersCount { get; }
      string CommandExchangeName { get; }
      ushort QueryPrefetchCount { get; }
      int QueryConsumersCount { get; }
      string QueryExchangeName { get; }
      string EventExchangeName { get; }
      ushort EventPrefetchCount { get; }
      int EventConsumersCount { get; }
   }
}
using Microsoft.Extensions.Configuration;
using IConfiguration =
RemoteScheduler.Core.Configuration.IConfiguration;
namespace RemoteScheduler.Core.Cqrs.RabbitMq.Configurations
{
   public class RabbitMqRpcConfiguration :
IRabbitMqRpcConfiguration, IConfiguration
   {
      public ushort CommandPrefetchCount { get; set; }
      public int CommandConsumersCount { get; set; }
      public string CommandExchangeName { get; set; }
      public ushort QueryPrefetchCount { get; set; }
      public int QueryConsumersCount { get; set; }
```

```
        public string QueryExchangeName { get; set; }
        public string EventExchangeName { get; set; }
        public ushort EventPrefetchCount { get; set; }
        public int EventConsumersCount { get; set; }
        public IConfigurationSection ConfigurationSection { get; set;
}
    }
}
namespace RemoteScheduler.Core.Cqrs.RabbitMq.Enums
{
    public enum ChannelTag
    {
        CommandPublish = 1,
        QueuePublish = 2,
        EventPublish = 3,
        RpcConsume = 4,
        CommandConsume = 5,
        QueryConsume = 6,
        EventConsume = 7,
        CommandResponsePublish = 8,
        QueryResponsePublish = 9
    }
}
namespace RemoteScheduler.Core.Cqrs.RabbitMq.Enums
{
    public enum ConnectionTag
    {
        Publish = 1,
        Consume = 2
    }
}
namespace RemoteScheduler.Core.Cqrs.RabbitMq.Exceptions
{
    public abstract class CqrsRabbitMqException
    {
    }
    public class RpcExecutionException
    {
    }
    public class RpcTimeoutException<TMessage> where
TMessage : class
    {
    }
}
using System.IO.Compression;
using System.Threading.Tasks;
using RabbitMQ.Client.Events;
namespace RemoteScheduler.Core.Cqrs.RabbitMq.Serialization
{
    public interface IMessageSerializer
    {
        byte[] Serialize(object message);
        TMessage Deserialize<TMessage>(byte[] rawMessage);
        object Deserialize(byte[] rawMessage);
    }
}
using RemoteScheduler.Core.Cqrs.RabbitMq.Serialization;
namespace RemoteScheduler.Core.Cqrs.RabbitMq.Server
{
    public class CommandRpcListener : RpcListenerBase
    {
        private readonly IRabbitMqRpcConfiguration _configuration;
        public CommandRpcListener(
            [KeyFilter(ChannelTag.CommandConsume)] IModel
consumeChannel,
            IRabbitMqRpcConfiguration configuration,
            ILifetimeScope lifetimeScope, IMessageSerializer
serializer) :
            base(consumeChannel, lifetimeScope, serializer)
        {
            _configuration = configuration;
        }
        protected override bool AutoAck => false;
        private readonly MessageExecutionContext
_executionContext = new MessageExecutionContext
        {
            ExecutionFilter = type =>
                type.IsGenericType &&
type.GetGenericTypeDefinition() !=
typeof(RpcCommandHandler<>)
        };
        private static readonly string AssemblyNamespace =
Constants.EntryAssemblyNamespace;
        protected override string ExchangeName =>
_configuration.CommandExchangeName;
        protected override string QueueName =>
$"{AssemblyNamespace}.CommandQueue";
        protected override string RoutingKey =>
$"{AssemblyNamespace}.#";
        protected override string ExchangeType =>
RabbitMQ.Client.ExchangeType.Topic;
        protected override ushort PrefetchCount =>
_configuration.CommandPrefetchCount;
        protected override int ConsumersCount =>
_configuration.CommandConsumersCount;
        protected override ChannelTag PublishChannelTag =>
ChannelTag.CommandResponsePublish;
        protected override async Task<object>
ProcessMessage(ILifetimeScope scope, BasicDeliverEventArgs
@event)
        {
            var commandBus = scope.Resolve<ICommandBus>();
            var command = Serializer.Deserialize(@event.Body);
            await commandBus.Execute((ICommand) command,
_executionContext);
            return new object();
        }
    }
}
using System;
using System.Collections.Generic;
using System.Reflection;
using System.Text;
using RemoteScheduler.Core.Cqrs.RabbitMq.Serialization;
namespace RemoteScheduler.Core.Cqrs.RabbitMq.Server
{
    public class EventRpcListener : RpcListenerBase
    {
        private readonly IRabbitMqRpcConfiguration _configuration;
        public EventRpcListener(
            [KeyFilter(ChannelTag.EventConsume)] IModel
consumeChannel,
            IRabbitMqRpcConfiguration configuration,
            ILifetimeScope lifetimeScope, IMessageSerializer
serializer) :
            base(consumeChannel, lifetimeScope, serializer)
        {
            _configuration = configuration;
        }
        protected override bool AutoAck => true;
        private readonly MessageExecutionContext
_executionContext = new MessageExecutionContext
        {
            ExecutionFilter = type =>
                !(type.IsGenericType &&
type.GetGenericTypeDefinition() ==
typeof(RpcEventSubscriber<>))
        };
        private static readonly string AssemblyNamespace =
Constants.EntryAssemblyNamespace;
        protected override string ExchangeName =>
_configuration.EventExchangeName;
        protected override string QueueName =>
$"{AssemblyNamespace}.EventQueue";
        protected override string RoutingKey =>
$"{AssemblyNamespace}.#";
        protected override string ExchangeType =>
RabbitMQ.Client.ExchangeType.Fanout;
        protected override ushort PrefetchCount =>
_configuration.EventPrefetchCount;
        protected override int ConsumersCount =>
_configuration.EventConsumersCount;
        protected override ChannelTag PublishChannelTag =>
ChannelTag.EventPublish;
```

```
        protected override async Task<object> ProcessMessage(ILifetimeScope scope, BasicDeliverEventArgs @event)
        {
            var sourceBytes = (byte[]) @event.BasicProperties.Headers["Source"];
            var eventSource = Encoding.UTF8.GetString(sourceBytes);
            if (eventSource == AssemblyNamespace)
            {
                return new object();
            }
            var eventBus = scope.Resolve<IEventBus>();
            var resolvedEvent = Serializer.Deserialize(@event.Body);
            await eventBus.Raise((IEvent) resolvedEvent, _executionContext);
            return new object();
        }
    }
}
using System;
using System.Collections.Generic;
using RemoteScheduler.Core.Cqrs.RabbitMq.Serialization;
namespace RemoteScheduler.Core.Cqrs.RabbitMq.Server
{
    public class QueryRpcListener : RpcListenerBase
    {
        private readonly IRabbitMqRpcConfiguration _configuration;
        public QueryRpcListener(
            [KeyFilter(ChannelTag.QueryConsume)] IModel consumeChannel,
            IRabbitMqRpcConfiguration configuration,
            ILifetimeScope lifetimeScope, IMessageSerializer serializer) :
            base(consumeChannel, lifetimeScope, serializer)
        {
            _configuration = configuration;
        }
        protected override bool AutoAck => false;
        private readonly MessageExecutionContext _executionContext = new MessageExecutionContext
        {
            ExecutionFilter = type =>
                type.IsGenericType && type.GetGenericTypeDefinition() != typeof(RpcQueryHandler<,>)
        };
        private static readonly string AssemblyNamespace = Constants.EntryAssemblyNamespace;
        protected override string ExchangeName => _configuration.QueryExchangeName;
        protected override string QueueName => $"{AssemblyNamespace}.QueryQueue";
        protected override string RoutingKey => $"{AssemblyNamespace}.#";
        protected override string ExchangeType => RabbitMQ.Client.ExchangeType.Topic;
        protected override ushort PrefetchCount => _configuration.QueryPrefetchCount;
        protected override int ConsumersCount => _configuration.QueryConsumersCount;
        protected override ChannelTag PublishChannelTag => ChannelTag.QueryResponsePublish;
        protected override async Task<object> ProcessMessage(ILifetimeScope scope, BasicDeliverEventArgs @event)
        {
            var queryBus = scope.Resolve<IQueryBus>();
            var query = Serializer.Deserialize(@event.Body);
            var response = await queryBus.Execute((IQuery) query, _executionContext);
            return response;
        }
    }
}
using System;
using System.Collections.Generic;
using RemoteScheduler.Core.Cqrs.RabbitMq.Exceptions;
using RemoteScheduler.Core.Cqrs.RabbitMq.Serialization;
namespace RemoteScheduler.Core.Cqrs.RabbitMq.Server
{
    public abstract class RpcListenerBase : IDisposable
    {
        private readonly IModel _consumeChannel;
        private readonly ILifetimeScope _rootLifetimeScope;
        protected readonly IMessageSerializer Serializer;
        private readonly ICollection<string> _consumerTags = new List<string>();
        protected abstract bool AutoAck { get; }
        protected abstract string ExchangeName { get; }
        protected abstract string QueueName { get; }
        protected abstract string RoutingKey { get; }
        protected abstract string ExchangeType { get; }
        protected abstract ushort PrefetchCount { get; }
        protected abstract int ConsumersCount { get; }
        protected abstract ChannelTag PublishChannelTag { get; }
        protected RpcListenerBase(IModel consumeChannel,
            ILifetimeScope rootLifetimeScope, IMessageSerializer serializer)
        {
            _consumeChannel = consumeChannel;
            _rootLifetimeScope = rootLifetimeScope;
            Serializer = serializer;
        }
        public void SetupListener()
        {
            _consumeChannel.ExchangeDeclare(ExchangeName, ExchangeType, true, false);
            _consumeChannel.QueueDeclare(QueueName, true, false, false);
            _consumeChannel.BasicQos(0, PrefetchCount, false);
            _consumeChannel.QueueBind(QueueName, ExchangeName, RoutingKey);
            for (int i = 0; i < ConsumersCount; i++)
            {
                var consumer = new AsyncEventingBasicConsumer(_consumeChannel);
                consumer.Received += ConsumerOnReceived;
                var tag = _consumeChannel.BasicConsume(QueueName, AutoAck, consumer);
                _consumerTags.Add(tag);
            }
        }
        private async Task ConsumerOnReceived(object sender, BasicDeliverEventArgs @event)
        {
            using (var scope = _rootLifetimeScope.BeginLifetimeScope("Rpc execution scope"))
            {
                try
                {
                    var response = await ProcessMessage(scope, @event);
                    if (!AutoAck)
                    {
                        SendResponse(scope, response, @event);
                    }
                }
                catch (Exception)
                {
                    if (!AutoAck)
                    {
                        SendResponse(scope, new ExceptionData(), @event);
                    }
                }
            }
        }
        protected abstract Task<object> ProcessMessage(ILifetimeScope scope, BasicDeliverEventArgs @event);
        private void SendResponse(ILifetimeScope scope, object response, BasicDeliverEventArgs eventArgs)
        {
            _consumeChannel.BasicAck(eventArgs.DeliveryTag, false);
            var publishChannel = scope.ResolveKeyed<IModel>(PublishChannelTag);
```

```
            var basicProperties =
publishChannel.CreateBasicProperties();
            basicProperties.CorrelationId =
eventArgs.BasicProperties.CorrelationId;
            var bytes = Serializer.Serialize(response);
            publishChannel.BasicPublish("",
eventArgs.BasicProperties.ReplyTo, basicProperties, bytes);
        }
        public void Dispose()
        {
            foreach (var tag in _consumerTags)
            {
                _consumeChannel?.BasicCancel(tag);
            }
            _consumeChannel?.Dispose();
        }
    }
}
using System;
namespace RemoteScheduler.Core.Exceptions
{
    public class EntityNotFoundException<TRequest, TEntity> :
RemoteSchedulerException
    {
        public Type EntityType => typeof(TEntity);
        public TRequest Request { get; }
        public EntityNotFoundException(TRequest request)
        {
            Request = request;
        }
    }
}
using System.Collections.Generic;
namespace RemoteScheduler.Core.Exceptions
{
    public static class ErrorMessages
    {
        public static IDictionary<string, string> Messages = new
Dictionary<string, string>
        {
            ["EntityNotFoundException"] = "Unable to find entity for
your request"
        };
        public static IDictionary<string, string> Codes = new
Dictionary<string, string>
        {
            ["EntityNotFoundException"] = "r.sched.core-0001"
        };
    }
}
using System;
using System.Collections
using Newtonsoft.Json.Linq;
using RemoteScheduler.Core.Exceptions.Attributes;
namespace RemoteScheduler.Core.Exceptions
{
    [StatusCode(HttpStatusCode.BadRequest)]
    public class RemoteSchedulerException : ApplicationException
    {
        [JsonIgnore] protected virtual IDictionary<string, string>
ExceptionCodes => ErrorMessages.Codes;
        [JsonIgnore] protected virtual IDictionary<string, string>
ExceptionMessages => ErrorMessages.Messages;
        public int StatusCode =>
GetType().GetCustomAttribute<StatusCodeAttribute>().StatusCod
e;
        public string ExceptionType =>
GetType().Name.Split('`').First();
        [Log, Response]
        public string Code => ExceptionCodes[ExceptionType];
        [Log, Response]
        public override string Message =>
ExceptionMessages[ExceptionType];
        public virtual JObject GetLogMessage()
        {
            return
GetObjectOfPropertiesWithAttribute<LogAttribute>();
        }
        public virtual JObject GetResponseMessage()
        {
            return
GetObjectOfPropertiesWithAttribute<ResponseAttribute>();
        }
        private JObject
GetObjectOfPropertiesWithAttribute<TAttribute>() where
TAttribute : Attribute
        {
            var logProperties =
GetType().GetProperties(BindingFlags.Public |
BindingFlags.Instance)
                .Where(info =>
info.GetCustomAttributes<TAttribute>().Any());
            var jObject = new JObject();
            var jConverters = new List<JsonConverter>
            {
                new StringEnumConverter()
            };
            var serializer = JsonSerializer.CreateDefault(new
JsonSerializerSettings
            {
                Converters = jConverters
            });
            foreach (var propertyInfo in logProperties)
            {
                var response = propertyInfo.GetMethod.Invoke(this,
parameters: null);
                var property = response != null
                    ? JToken.FromObject(response, serializer)
                    : new JValue((object) null);
                jObject.Add(propertyInfo.Name, property);
            }
            return jObject;
        }
    }
}
using System;
namespace RemoteScheduler.Core.Exceptions.Attributes
{
    [AttributeUsage(AttributeTargets.Property)]
    public class LogAttribute : Attribute
    {
    }
}
using System;
namespace RemoteScheduler.Core.Exceptions.Attributes
{
    [AttributeUsage(AttributeTargets.Property)]
    public class ResponseAttribute : Attribute
    {
    }
}
using System;
using System.Net;
namespace RemoteScheduler.Core.Exceptions.Attributes
{
    [AttributeUsage(AttributeTargets.Class, Inherited = true)]
    public class StatusCodeAttribute : Attribute
    {
        public int StatusCode { get; }
        public StatusCodeAttribute(HttpStatusCode statusCode)
        {
            StatusCode = (int) statusCode;
        }
        public StatusCodeAttribute(int statusCode)
        {
            StatusCode = statusCode;
        }
    }
}
using System.IO;
using log4net.ObjectRenderer;
namespace RemoteScheduler.Core.Logging
{
    public class JObjectRenderer : IObjectRenderer
```

```
    {
        public void RenderObject(RendererMap rendererMap, object
obj, TextWriter writer)
        {
            writer.Write(obj.ToString());
        }
    }
}
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
namespace RemoteScheduler.Services.Api
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateWebHostBuilder(args).Build().Run();
        }
        public static IWebHostBuilder CreateWebHostBuilder(string[]
args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>();
    }
}
using System;
using System.Reflection;
using Swashbuckle.AspNetCore.SwaggerUI;
using                    Environment                    =
RemoteScheduler.Core.Configuration.Environment;
namespace RemoteScheduler.Services.Api
{
    public class Startup
    {
        private static readonly IEnvironment Environment = new
Environment();
        public IServiceProvider ConfigureServices(IServiceCollection
services)
        {
            services.AddSwaggerGen(options =>
            {
                options.DescribeAllEnumsAsStrings();
                options.OperationFilter<AuthorizedOperationFilter>();
                options.SwaggerDoc("v1", new Info
                {
                    Title = "Remote Scheduler API",
                    Version = "v1"
                });
                options.DescribeAllParametersInCamelCase();
                options.DescribeStringEnumsInCamelCase();
            });
            services.AddMvc()
                .AddJsonOptions(options =>
                {
                    options.SerializerSettings.ContractResolver = new
CamelCasePropertyNamesContractResolver();
                    options.SerializerSettings.Converters.Add(new
StringEnumConverter());
                    options.SerializerSettings.MissingMemberHandling =
MissingMemberHandling.Error;
                })
                .AddMvcOptions(options                    =>                    {
options.Filters.Add<ExceptionsFilter>(); });
            var builder = new ContainerBuilder();
            builder.Populate(services);
            builder.RegisterModule<CqrsModule>();
            builder.RegisterModule<CqrsRmqModule>();
            builder.RegisterModule<EnvironmentModule>();
            builder.RegisterModule<Log4NetConfigurationModule>();
            builder.RegisterModule<Log4NetProviderModule>();
            builder.RegisterModule<ApiImplementationModule>();
            builder.RegisterModule<ApiDatabaseModule>();

builder.RegisterConfiguration(Environment.EnvironmentName,
                Assembly.GetExecutingAssembly(),
                typeof(ILog4NetConfiguration).Assembly,
                typeof(RabbitMqRpcConfiguration).Assembly);
            builder.Register(context =>
            {
                var                rabbitMqConfiguration                =
context.Resolve<IRabbitMqConfiguration>();
                var connectionFactory = new ConnectionFactory
                {
                    DispatchConsumersAsync = true,
                    HostName = rabbitMqConfiguration.Host,
                    UserName = rabbitMqConfiguration.UserName,
                    Password = rabbitMqConfiguration.Password,
                    VirtualHost = rabbitMqConfiguration.VirtualHost,
                    AutomaticRecoveryEnabled                =
rabbitMqConfiguration.AutoRecovery,
                    Port = 5672
                };
                return connectionFactory;
            })
            .AsImplementedInterfaces()
            .SingleInstance();
            builder.RegisterType<BearerSchemeProcessor>()

.Keyed<IAuthorizationSchemeProcessor>(AuthorizationScheme.B
earer);
            builder.RegisterType<ApiKeySchemeProcessor>()

.Keyed<IAuthorizationSchemeProcessor>(AuthorizationScheme.A
piKey);
            var container = builder.Build();
            return new AutofacServiceProvider(container);
        }
        public void Configure(IApplicationBuilder app,
IHostingEnvironment env)
        {
            app.UseForwardedHeaders(new ForwardedHeadersOptions
            {
                ForwardedHeaders = ForwardedHeaders.All
            });
            app.UseSwagger();
            app.UseSwaggerUI(options =>
            {
                options.RoutePrefix = string.Empty;
                options.ConfigObject.DocExpansion                =
DocExpansion.None;
                options.DocumentTitle = "RemoteScheduler.Api";
                options.SwaggerEndpoint("/swagger/v1/swagger.json",
"RemoteScheduler.Api");
                options.DisplayRequestDuration();
            });
            app.UseMvc();
        }
    }
}
using System;
using Microsoft.Extensions.Configuration;
using RemoteScheduler.Services.Api.Abstractions.Configurations;
using                    IConfiguration                    =
RemoteScheduler.Core.Configuration.IConfiguration;
namespace RemoteScheduler.Services.Api.Configurations
{
    public          class          AccessTokenConfiguration          :
IAccessTokenConfiguration, IConfiguration
    {
        public string SecretKey { get; set; }
        public TimeSpan Lifetime { get; set; }
        public IConfigurationSection ConfigurationSection { get; set;
}
    }
}
using Microsoft.Extensions.Configuration;
using RemoteScheduler.Services.Api.Abstractions.Configurations;
using                    IConfiguration                    =
RemoteScheduler.Core.Configuration.IConfiguration;
namespace RemoteScheduler.Services.Api.Configurations
{
    public class DomainsConfiguration : IDomainsConfiguration,
IConfiguration
    {
        public string ApiDomain { get; set; }
```

```
        public IConfigurationSection ConfigurationSection { get; set;
}
    }
}
using Microsoft.Extensions.Configuration;
using RemoteScheduler.Services.Api.Abstractions.Configurations;
namespace RemoteScheduler.Services.Api.Configurations
{
    public class RabbitMqConfiguration : IRabbitMqConfiguration
    {
        public IConfigurationSection ConfigurationSection { get; set;
}
        public string Host { get; set; }
        public string UserName { get; set; }
        public string Password { get; set; }
        public string VirtualHost { get; set; }
        public bool AutoRecovery { get; set; }
    }
}
using Microsoft.AspNetCore.Mvc;
using RemoteScheduler.Core.Cqrs.Abstractions.Buses;
namespace RemoteScheduler.Services.Api.Controllers.Abstract
{
    public abstract class BusController : ControllerBase
    {
        protected ICommandBus CommandBus { get; }
        protected IQueryBus QueryBus { get; }
        protected BusController(ICommandBus commandBus,
IQueryBus queryBus)
        {
            CommandBus = commandBus;
            QueryBus = queryBus;
        }
    }
}
using System;
using RemoteScheduler.Services.Api.Security;
namespace RemoteScheduler.Services.Api.Controllers.Abstract
{
    public class SecuredController : BusController,
IAsyncActionFilter
    {
        private readonly IIndex<AuthorizationScheme,
IAuthorizationSchemeProcessor> _processorsIndex;
        public SecuredController(ICommandBus commandBus,
IQueryBus queryBus,
            IIndex<AuthorizationScheme,
IAuthorizationSchemeProcessor> processorsIndex) :
base(commandBus, queryBus)
        {
            _processorsIndex = processorsIndex;
        }
        protected Guid UserId { get; private set; }
        [NonAction]
        public async Task
OnActionExecutionAsync(ActionExecutingContext context,
ActionExecutionDelegate next)
        {
            if (!Request.Headers.TryGetValue("Authorization", out var
headers))
            {
                throw new AuthorizationHeaderMissingException();
            }
            var authHeader =
AuthenticationHeaderValue.Parse(headers.First());
            var authScheme = authHeader.Scheme;
            if (!(Enum.TryParse(authScheme, out AuthorizationScheme
scheme) &&
                _processorsIndex.TryGetValue(scheme, out var
processor)))
            {
                throw new UnknownAuthorizationSchemeException();
            }
            var userPayload = await
processor.Authorize<UserPayload>(Request,
authHeader.Parameter);
            UserId = userPayload.UserId;
            await next.Invoke();
        }
    }
}
using System;
using RemoteScheduler.Services.Api.Controllers.Public.Models;
using RemoteScheduler.Services.Api.Security;
namespace RemoteScheduler.Services.Api.Controllers.Public
{
    [Route("api/public/login")]
    public class PublicLoginController : BusController
    {
        private readonly IAccessTokenService _accessTokenService;
        private readonly IRandomStringGenerationService
_randomStringGenerationService;
        private readonly ISha512HashingService
_sha512HashingService;
        private readonly IDomainsConfiguration
_domainsConfiguration;
        public PublicLoginController(ICommandBus commandBus,
IQueryBus queryBus, IAccessTokenService accessTokenService,
            IRandomStringGenerationService
randomStringGenerationService, ISha512HashingService
sha512HashingService,
            IDomainsConfiguration domainsConfiguration) :
            base(commandBus, queryBus)
        {
            _accessTokenService = accessTokenService;
            _randomStringGenerationService =
randomStringGenerationService;
            _sha512HashingService = sha512HashingService;
            _domainsConfiguration = domainsConfiguration;
        }
        [HttpPost]
        public async Task<LoginResponseModel> Login([FromBody]
LoginRequestModel loginRequestModel)
        {
            var query = new UserByCredentialsQuery
            {
                Email = loginRequestModel.Email,
                PasswordHash =
_sha512HashingService.GenerateHash(loginRequestModel.Passwo
rd)
            };
            var userResponse = await
QueryBus.Execute<UserByCredentialsQuery,
UserByCredentialsResponse>(query);
            var refreshToken =
_randomStringGenerationService.Generate(64);
            var accessToken = _accessTokenService.CreateToken(new
UserPayload
            {
                UserId = userResponse.UserId
            }, refreshToken);
            var deviceId = GetOrCreateDeviceId();
            var command = new SetUserRefreshTokenCommand
            {
                DeviceId = deviceId,
                UserId = userResponse.UserId,
                RefreshToken = refreshToken
            };
            await CommandBus.Execute(command);
            SetDeviceIdCookie(deviceId);
            return new LoginResponseModel
            {
                AccessToken = accessToken,
                RefreshToken = refreshToken
            };
        }
        [HttpPost("refresh-token")]
        public async Task<RefreshTokenResponseModel>
RefreshToken([FromBody] RefreshTokenRequestModel model)
        {
            if (!TryGetDeviceId(out var deviceId))
            {
                throw new DeviceIdMissingException();
            }
```

```
            var query = new UserByRefreshTokenQuery
            {
                DeviceId = deviceId,
                RefreshToken = model.RefreshToken
            };
            var userInfo = await
QueryBus.Execute<UserByRefreshTokenQuery,
UserByRefreshTokenResponse>(query);
            var accessToken = _accessTokenService.CreateToken(new
UserPayload
            {
                UserId = userInfo.UserId
            }, model.RefreshToken);
            return new RefreshTokenResponseModel
            {
                AccessToken = accessToken
            };
        }
        private void SetDeviceIdCookie(Guid deviceId)
        {
            Response.Cookies.Append("DeviceId", deviceId.ToString(),
new CookieOptions
            {
                Domain = _domainsConfiguration.ApiDomain,
                Path = "/"
            });
        }
        private bool TryGetDeviceId(out Guid deviceId)
        {
            if (!Request.Cookies.TryGetValue("DeviceId", out var
deviceIdString))
            {
                deviceId = Guid.Empty;
                return false;
            }
            if (!Guid.TryParse(deviceIdString, out deviceId))
            {
                return false;
            }
            return true;
        }
        private Guid GetOrCreateDeviceId()
        {
            if (!Request.Cookies.TryGetValue("DeviceId", out var
deviceIdString))
            {
                return Guid.NewGuid();
            }
            if (!Guid.TryParse(deviceIdString, out var deviceId))
            {
                return Guid.NewGuid();
            }
            return deviceId;
        }
    }
}
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using RemoteScheduler.Services.Api.Controllers.Public.Models;
namespace RemoteScheduler.Services.Api.Controllers.Public
{
    [Route("api/public/register")]
    public class PublicRegistrationController : BusController
    {
        private readonly ISha512HashingService
_sha512HashingService;
        public PublicRegistrationController(ICommandBus
commandBus, IQueryBus queryBus,
            ISha512HashingService sha512HashingService) :
base(commandBus, queryBus)
        {
            _sha512HashingService = sha512HashingService;
        }
        [HttpPost]
        public async Task Register([FromBody]
RegistrationRequestModel model)
        {
            var generateHash =
_sha512HashingService.GenerateHash(model.Password);
            var registerUserCommand = new RegisterUserCommand
            {
                Email = model.Email?.ToLowerInvariant(),
                PasswordHash = generateHash,
                FirstName = model.FirstName,
                LastName = model.LastName
            };
            await CommandBus.Execute(registerUserCommand);
        }
    }
}
namespace RemoteScheduler.Services.Api.Controllers.Public
{
    public class PublicResetPasswordController
    {
    }
}
namespace
RemoteScheduler.Services.Api.Controllers.Public.Models
{
    public class LoginRequestModel
    {
        public string Email { get; set; }
        public string Password { get; set; }
    }
}
namespace
RemoteScheduler.Services.Api.Controllers.Public.Models
{
    public class LoginResponseModel
    {
        public string AccessToken { get; set; }
        public string RefreshToken { get; set; }
    }
}
namespace
RemoteScheduler.Services.Api.Controllers.Public.Models
{
    public class RefreshTokenRequestModel
    {
        public string RefreshToken { get; set; }
    }
}
namespace
RemoteScheduler.Services.Api.Controllers.Public.Models
{
    public class RefreshTokenResponseModel
    {
        public string AccessToken { get; set; }
    }
}
namespace
RemoteScheduler.Services.Api.Controllers.Public.Models
{
    public class RegistrationRequestModel
    {
        public string Email { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Password { get; set; }
    }
}
using System.Threading.Tasks;
using Autofac.Features.Indexed;
using RemoteScheduler.Services.Api.Controllers.User.Models;
using RemoteScheduler.Services.Api.Security;
namespace RemoteScheduler.Services.Api.Controllers.User
{
    [Route("api/user/details")]
    public class UserDetailsController : SecuredController
    {
        private readonly ISha512HashingService
_sha512HashingService;
        private readonly IRandomStringGenerationService
_randomStringGenerationService;
```

```
        public UserDetailsController(ICommandBus commandBus,
IQueryBus queryBus,
            IIndex<AuthorizationScheme,
IAuthorizationSchemeProcessor> processorsIndex,
            ISha512HashingService sha512HashingService,
IRandomStringGenerationService randomStringGenerationService)
: base(commandBus, queryBus,
            processorsIndex)
        {
            _sha512HashingService = sha512HashingService;
            _randomStringGenerationService =
randomStringGenerationService;
        }
        [HttpGet]
        public async Task<UserDetailsResponse> GetUserDetails()
        {
            var userDetailsResponse = await
QueryBus.Execute<UserDetailsQuery, UserDetailsResponse>(new
UserDetailsQuery
            {
                UserId = UserId
            });
            return userDetailsResponse;
        }
        [HttpPatch("details")]
        public async Task UpdateUserDetails([FromBody]
UpdateUserDetailsModel model)
        {
            var command = new UpdateUserDetailsCommand
            {
                UserId = UserId,
                FirstName = model?.FirstName,
                LastName = model?.LastName,
                Email = model?.Email?.ToLowerInvariant()
            };
            await CommandBus.Execute(command);
        }
        [HttpPatch("upsert-allowed-ips")]
        public async Task UpdateAllowedIps([FromBody]
UpserAllowedIpsModel model)
        {
            var upsertAllowedIpsCommand = new
UpsertAllowedIpsCommand
            {
                UserId = UserId,
                AllowedIps = model.AllowedIps
            };
            await CommandBus.Execute(upsertAllowedIpsCommand);
        }
        [HttpPatch("recreate-api-key")]
        public async Task RecreateApiKey()
        {
            var updateApiKeyCommand = new
UpdateApiKeyCommand
            {
                ApiKey =
_randomStringGenerationService.Generate(96),
                UserId = UserId,
            };
            await CommandBus.Execute(updateApiKeyCommand);
        }
        [HttpPatch("password")]
        public async Task ChangeUserPassword([FromBody]
ChangePasswordModel model)
        {
            if (string.IsNullOrEmpty(model.Password))
            {
                throw new UserByCredentialsNotFoundException();
            }
            var changePasswordCommand = new
ChangePasswordCommand
            {
                UserId = UserId,
                PasswordHash =
_sha512HashingService.GenerateHash(model.Password)
            };
            await CommandBus.Execute(changePasswordCommand);
        }
    }
}
using System;
using System.Threading.Tasks;
using
RemoteScheduler.Services.Notifications.Email.Contracts.Comman
ds;
using RemoteScheduler.Services.Scheduling.Contracts.Commands;
namespace RemoteScheduler.Services.Api.Controllers.User.Email
{
    [Route("api/user/notifications/email")]
    public class UserEmailNotificationsController :
SecuredController
    {
        public UserEmailNotificationsController(ICommandBus
commandBus, IQueryBus queryBus,
            IIndex<AuthorizationScheme,
IAuthorizationSchemeProcessor> processorsIndex) :
base(commandBus, queryBus,
            processorsIndex)
        {
        }
        [HttpPost("schedule")]
        public async Task ScheduleNewNotification([FromBody]
CreateEmailNotificationModel model)
        {
            var scheduleCommandExecutionCommand = new
ScheduleCommandExecutionCommand
            {
                Interval = model.Interval,
                RepeatsCount = model.RepeatsCount,
                JobId = Guid.NewGuid(),
                StartAt = model.StartAt,
                Command = new SendEmailNotificationCommand
                {
                    Subject = "asd",
                    EmailSettingsId = null
                }
            };
            await
CommandBus.Execute(scheduleCommandExecutionCommand);
        }
        [HttpPost("schedule/cron")]
        public async Task
ScheduleNewNotificationWithCron([FromBody]
CreateHttpNotificationWithCronModel model)
        {
        }
        [HttpPut("{notificationId}")]
        public async Task UpdateScheduledNotification([FromRoute]
Guid notificationId,
            [FromBody] UpdateHttpNotificationModel model)
        {
        }
        [HttpDelete("cancel/{notificationId}")]
        public async Task CancelScheduledNotification([FromRoute]
Guid notificationId)
        {
        }
        [HttpGet]
        public async Task<HttpNotificationsResponse>
GetHttpNotifications()
        {
            return new HttpNotificationsResponse();
        }
    }
}
using System.Threading.Tasks;
using Autofac.Features.Indexed;
using Microsoft.AspNetCore.Mvc;
using RemoteScheduler.Core.Cqrs.Abstractions.Buses;
using RemoteScheduler.Services.Api.Security;
namespace RemoteScheduler.Services.Api.Controllers.User.Email
{
    [Route("api/user/email/receivers")]
    public class UserEmailReceiversController : SecuredController
```

```
    {
        public UserEmailReceiversController(ICommandBus
commandBus, IQueryBus queryBus,
            IIndex<AuthorizationScheme,
IAuthorizationSchemeProcessor> processorsIndex) :
base(commandBus, queryBus,
            processorsIndex)
        {
        }
        [HttpPost]
        public async Task AddReceiver()
        {
        }
        [HttpPut("{receiverId}")]
        public async Task UpdateReceiver()
        {
        }
        [HttpDelete("{receiverId}")]
        public async Task RemoveReceiver()
        {
        }
        [HttpGet]
        public async Task GetReceivers()
        {
        }
    }
}
using System.Threading.Tasks;
using Autofac.Features.Indexed;
using Microsoft.AspNetCore.Mvc;
using RemoteScheduler.Core.Cqrs.Abstractions.Buses;
using
RemoteScheduler.Services.Api.Abstractions.Services.Authorization
;
using RemoteScheduler.Services.Api.Controllers.Abstract;
using RemoteScheduler.Services.Api.Security;
namespace RemoteScheduler.Services.Api.Controllers.User.Email
{
    [Route("api/user/email/settings")]
    public class UserEmailSettingsController : SecuredController
    {
        public UserEmailSettingsController(ICommandBus
commandBus, IQueryBus queryBus,
            IIndex<AuthorizationScheme,
IAuthorizationSchemeProcessor> processorsIndex) :
base(commandBus, queryBus,
            processorsIndex)
        {
        }
        [HttpPost]
        public async Task AddSettings()
        {
        }
        [HttpPut("{settingId}")]
        public async Task UpdateSettings()
        {
        }
        [HttpDelete("{settingId}")]
        public async Task RemoveSettings()
        {
        }
        [HttpGet]
        public async Task GetSettings()
        {
        }
    }
}
using System.Threading.Tasks;
using Autofac.Features.Indexed;
using Microsoft.AspNetCore.Mvc;
using RemoteScheduler.Core.Cqrs.Abstractions.Buses;
using
RemoteScheduler.Services.Api.Abstractions.Services.Authorization
;
using RemoteScheduler.Services.Api.Controllers.Abstract;
using RemoteScheduler.Services.Api.Security;
namespace RemoteScheduler.Services.Api.Controllers.User.Email
{
    [Route("api/user/email/templates")]
    public class UserEmailTemplatesController : SecuredController
    {
        [HttpPost]
        public async Task AddTemplate()
        {
        }
        [HttpPut("{templateId}")]
        public async Task UpdateTemplate()
        {
        }
        [HttpDelete("{templateId}")]
        public async Task RemoveTemplate()
        {
        }
        [HttpGet("list")]
        public async Task GetTemplates()
        {
        }
        [HttpGet("details/{templateId}")]
        public async Task GetTemplateById()
        {
        }
        public UserEmailTemplatesController(ICommandBus
commandBus, IQueryBus queryBus, IIndex<AuthorizationScheme,
IAuthorizationSchemeProcessor> processorsIndex) :
base(commandBus, queryBus, processorsIndex)
        {
        }
    }
}
using System;
using System.Collections.Generic;
using Newtonsoft.Json.Linq;
using RemoteScheduler.Shared.Contracts.Enums;
namespace
RemoteScheduler.Services.Api.Controllers.User.Email.Models
{
    public class CreateEmailNotificationModel
    {
        public DateTime StartAt { get; set; }
        public int RepeatsCount { get; set; }
        public TimeSpan? Interval { get; set; }
        public ICollection<Guid> ReceiversIds { get; set; }
        public Guid EmailTemplateId { get; set; }
        public Guid? EmailSettingsId { get; set; }
        public string Subject { get; set; }
    }
}
using System;
using System.Threading.Tasks;
using Autofac.Features.Indexed;;
namespace RemoteScheduler.Services.Api.Controllers.User.Http
{
    [Route("api/user/notifications/http")]
    public class UserHttpNotificationsController : SecuredController
    {
        public UserHttpNotificationsController(ICommandBus
commandBus, IQueryBus queryBus,
            IIndex<AuthorizationScheme,
IAuthorizationSchemeProcessor> processorsIndex) :
base(commandBus, queryBus,
            processorsIndex)
        {
        }
        [HttpPost("schedule")]
        public async Task ScheduleNewNotification([FromBody]
CreateHttpNotificationModel model)
        {
            var createHttpNotificationCommand = new
CreateHttpNotificationCommand
            {
                Body = model.Body,
                Domain = model.Domain,
                Headers = model.Headers,
                Interval = model.Interval,
```

```
            Path = model.Path,
            Scheme = model.Scheme,
            ActionType = model.ActionType,
            CronSchedule = null,
            QueryParameters = model.QueryParameters,
            RepeatsCount = model.RepeatsCount - 1,
            StartAt = model.StartAt,
            UserId = UserId
        };
        await
CommandBus.Execute(createHttpNotificationCommand);
    }
    [HttpPost("schedule/cron")]
    public async Task
ScheduleNewNotificationWithCron([FromBody]
CreateHttpNotificationWithCronModel model)
    {
        var createHttpNotificationCommand = new
CreateHttpNotificationCommand
        {
            Body = model.Body,
            Domain = model.Domain,
            Headers = model.Headers,
            CronSchedule = model.CronExpression,
            Path = model.Path,
            Scheme = model.Scheme,
            ActionType = model.ActionType,
            QueryParameters = model.QueryParameters,
            StartAt = model.StartAt,
            UserId = UserId
        };
        await
CommandBus.Execute(createHttpNotificationCommand);
    }
    [HttpPut("{notificationId}")]
    public async Task UpdateScheduledNotification([FromRoute]
Guid notificationId,
        [FromBody] UpdateHttpNotificationModel model)
    {
        var updateHttpNotificationCommand = new
UpdateHttpNotificationCommand
        {
            UserId = UserId,
            NotificationId = notificationId,
            Body = model.Body,
            Domain = model.Domain,
            Headers = model.Headers,
            Path = model.Path,
            Scheme = model.Scheme,
            ActionType = model.ActionType,
            QueryParameters = model.QueryParameters
        };
        await
CommandBus.Execute(updateHttpNotificationCommand);
    }
    [HttpDelete("cancel/{notificationId}")]
    public async Task CancelScheduledNotification([FromRoute]
Guid notificationId)
    {
        var cancelHttpNotificationCommand = new
CancelHttpNotificationCommand
        {
            NotificationId = notificationId,
            UserId = UserId
        };
        await
CommandBus.Execute(cancelHttpNotificationCommand);
    }
    [HttpGet]
    public async Task<HttpNotificationsResponse>
GetHttpNotifications()
    {
        var httpNotificationsQuery = new HttpNotificationsQuery
        {
            UserId = UserId
        };
        var httpNotificationsResponse =
            await QueryBus.Execute<HttpNotificationsQuery,
HttpNotificationsResponse>(httpNotificationsQuery);
        return httpNotificationsResponse;
    }
  }
}
using System;
using System.Collections.Generic;
using Newtonsoft.Json.Linq;
using RemoteScheduler.Shared.Contracts.Enums;
namespace
RemoteScheduler.Services.Api.Controllers.User.Http.Models
{
  public class CreateHttpNotificationModel
  {
    public DateTime StartAt { get; set; }
    public int RepeatsCount { get; set; }
    public TimeSpan? Interval { get; set; }
    public string Scheme { get; set; }
    public string Domain { get; set; }
    public string Path { get; set; }
    public Dictionary<string, string> Headers { get; set; }
    public Dictionary<string, string> QueryParameters { get; set; }
    public HttpActionType ActionType{ get; set; }
    public JObject Body { get; set; }
  }
}
using System;
using System.Collections.Generic;
using Newtonsoft.Json.Linq;
using RemoteScheduler.Shared.Contracts.Enums;
namespace
RemoteScheduler.Services.Api.Controllers.User.Http.Models
{
  public class CreateHttpNotificationWithCronModel
  {
    public DateTime StartAt { get; set; }
    public string CronExpression { get; set; }
    public string Scheme { get; set; }
    public string Domain { get; set; }
    public string Path { get; set; }
    public Dictionary<string, string> Headers { get; set; }
    public Dictionary<string, string> QueryParameters { get; set; }
    public HttpActionType ActionType{ get; set; }
    public JObject Body { get; set; }
  }
}
using System.Collections.Generic;
using Newtonsoft.Json.Linq;
using RemoteScheduler.Shared.Contracts.Enums;
namespace
RemoteScheduler.Services.Api.Controllers.User.Http.Models
{
  public class UpdateHttpNotificationModel
  {
    public string Scheme { get; set; }
    public string Domain { get; set; }
    public string Path { get; set; }
    public Dictionary<string, string> Headers { get; set; }
    public Dictionary<string, string> QueryParameters { get; set; }
    public HttpActionType ActionType{ get; set; }
    public JObject Body { get; set; }
  }
}
namespace RemoteScheduler.Services.Api.Controllers.User.Models
{
  public class ChangePasswordModel
  {
    public string Password { get; set; }
  }
}
namespace RemoteScheduler.Services.Api.Controllers.User.Models
{
  public class UpdateUserDetailsModel
  {
    public string FirstName { get; set; }
    public string LastName { get; set; }
```

```
        public string Email { get; set; }
    }
}
namespace RemoteScheduler.Services.Api.Controllers.User.Models
{
    public class UpserAllowedIpsModel
    {
        public string[] AllowedIps { get; set; }
    }
}
using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http.Extensions;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json.Linq;
using RemoteScheduler.Core.Exceptions;
using RemoteScheduler.Core.Logging;
namespace RemoteScheduler.Services.Api.Filters
{
    public class ExceptionsFilter : IAsyncExceptionFilter
    {
        private readonly ILogger<ExceptionsFilter> _logger;
        public ExceptionsFilter(ILogger<ExceptionsFilter> logger)
        {
            _logger = logger;
        }
        public async Task OnExceptionAsync(ExceptionContext context)
        {
            var responseObject = new JObject
            {
                ["ApiRequestId"] = context.HttpContext.TraceIdentifier
            };
            if (context.Exception is RemoteSchedulerException schedulerException)
            {
                var responseMessage = schedulerException.GetResponseMessage();
                var statusCode = schedulerException.StatusCode;
                await Log(context);
                responseObject.Merge(responseMessage, new JsonMergeSettings
                    {
                        MergeArrayHandling = MergeArrayHandling.Union,
                        MergeNullValueHandling = MergeNullValueHandling.Merge
                    }
                );
                context.Result = new JsonResult(responseObject)
                {
                    StatusCode = statusCode
                };
            }
            else
            {
                responseObject.Add("Message", "Internal error, contact our support with Id of your request");
                await Log(context);
                context.Result = new JsonResult(responseObject)
                {
                    StatusCode = 500
                };
            }
        }
        private async Task Log(ExceptionContext context)
        {
            var requestId = context.HttpContext.TraceIdentifier;
            try
            {
                var request = context.HttpContext.Request;
                var content = request.Body;
                var contentString = await ReadRequestBody(content);
                var exceptionObject = new JObject
                {
                    ["RequestUrl"] = context.HttpContext.Request.GetDisplayUrl(),
                    ["Headers"] = JToken.FromObject(request.Headers),
                    ["Body"] = contentString,
                    ["Method"] = request.Method,
                    ["RequestId"] = requestId
                };
                if (context.Exception is RemoteSchedulerException)
                {
                    _logger.Warn(exceptionObject, context.Exception);
                }
                else
                {
                    _logger.Error(exceptionObject, context.Exception);
                }
            }
            catch (Exception e)
            {
                _logger.Fatal(new JObject
                {
                    ["Message"] = $"Exception log serialization error! RequestId {requestId}"
                }, e);
            }
        }
        private async Task<string> ReadRequestBody(Stream content)
        {
            string contentString;
            if (content.CanSeek && content.CanRead)
            {
                content.Seek(offset: 0, origin: SeekOrigin.Begin);
                var contentReader = new StreamReader(content);
                if (content.Length > 8092)
                {
                    var buffer = new char[8092];
                    await contentReader.ReadAsync(buffer, 0, buffer.Length);
                    contentString = new JObject
                    {
                        ["AdditionalMessage"] =
                            $"Content too long. Max content length {8092}. Actual length: {content.Length}",
                        ["Content"] = new string(buffer)
                    }.ToString();
                }
                else
                {
                    contentString = await contentReader.ReadToEndAsync();
                }
            }
            else
            {
                contentString = "Cannot read content from Body because it is too long or unavailable";
            }
            return contentString;
        }
    }
}
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using RemoteScheduler.Core.Cqrs.Abstractions.Buses;
using RemoteScheduler.Services.Api.Abstractions.Exceptions.Authorization;
using RemoteScheduler.Services.Api.Abstractions.Queries.Authorization;
using RemoteScheduler.Services.Api.Abstractions.Services.Authorization;
namespace RemoteScheduler.Services.Api.Security
{
```

```
public class ApiKeySchemeProcessor : IAuthorizationSchemeProcessor
{
    private readonly IQueryBus _queryBus;
    public ApiKeySchemeProcessor(IQueryBus queryBus)
    {
        _queryBus = queryBus;
    }
    public async Task<TPayload> Authorize<TPayload>(HttpRequest httpRequest, string token)
        where TPayload : IAccessTokenPayload, new()
    {
        var response = await _queryBus.Execute<UserApiKeyInfoByIdQuery, UserApiKeyInfoByIdResponse>(
            new UserApiKeyInfoByIdQuery
            {
                ApiKey = token
            });
        var remoteIpAddress = httpRequest.HttpContext.Connection.RemoteIpAddress;
        if (!response.AllowedIps.Contains(remoteIpAddress.ToString()))
        {
            throw new IpAddressIsNotAuthorizedException(remoteIpAddress.ToString(), response.AllowedIps);
        }
        return new TPayload
        {
            UserId = response.UserId
        };
    }
}
}
namespace RemoteScheduler.Services.Api.Security
{
    public enum AuthorizationScheme
    {
        ApiKey = 1,
        Bearer = 2
    }
}
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using RemoteScheduler.Core.Cqrs.Abstractions.Buses;
using RemoteScheduler.Services.Api.Abstractions.Exceptions.Authorization;
using RemoteScheduler.Services.Api.Abstractions.Queries.Authorization;
using RemoteScheduler.Services.Api.Abstractions.Services.Authorization;
namespace RemoteScheduler.Services.Api.Security
{
    public class BearerSchemeProcessor : IAuthorizationSchemeProcessor
    {
        private readonly IQueryBus _queryBus;
        private readonly IAccessTokenService _accessTokenService;
        public BearerSchemeProcessor(IQueryBus queryBus, IAccessTokenService accessTokenService)
        {
            _queryBus = queryBus;
            _accessTokenService = accessTokenService;
        }
        public async Task<TPayload> Authorize<TPayload>(HttpRequest httpRequest, string token)
            where TPayload : IAccessTokenPayload, new()
        {
            if (!httpRequest.Cookies.TryGetValue("DeviceId", out var deviceIdString))
            {
                throw new DeviceIdMissingException();
            }
            if (!Guid.TryParse(deviceIdString, out var deviceId))
            {
                throw new DeviceIdMissingException();
            }
            var accessTokenPayload = _accessTokenService.GetPayload<TPayload>(token);
            var response = await _queryBus.Execute<UserRefreshTokenByIdQuery, UserRefreshTokenByIdResponse>(
                new UserRefreshTokenByIdQuery
                {
                    UserId = accessTokenPayload.UserId,
                    DeviceId = deviceId
                });
            _accessTokenService.ValidateToken<TPayload>(token, response.RefreshToken);
            return accessTokenPayload;
        }
    }
}
using System;
using RemoteScheduler.Services.Api.Abstractions.Services.Authorization;
namespace RemoteScheduler.Services.Api.Security
{
    public class UserPayload : IAccessTokenPayload
    {
        public Guid UserId { get; set; }
        public DateTime ExpirationDate { get; set; }
        public long Lifetime { get; set; }
    }
}
using System.Collections.Generic;
using Autofac;
using Microsoft.AspNetCore.Mvc.Controllers;
using RemoteScheduler.Services.Api.Controllers.Abstract;
using Swashbuckle.AspNetCore.Swagger;
using Swashbuckle.AspNetCore.SwaggerGen;
namespace RemoteScheduler.Services.Api.Swagger
{
    public class AuthorizedOperationFilter : IOperationFilter
    {
        public void Apply(Operation operation, OperationFilterContext context)
        {
            if (!((ControllerActionDescriptor)(context.ApiDescription.ActionDescriptor)).ControllerTypeInfo.AsType()
                .IsAssignableTo<SecuredController>())
            {
                return;
            }
            if (operation.Parameters == null)
                operation.Parameters = new List<IParameter>();
            operation.Parameters.Add(new NonBodyParameter
            {
                Name = "Authorization",
                In = "header",
                Description = "Auth header",
                Required = true,
                Type = "string",
                Default = "Bearer"
            });
        }
    }
}
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace RemoteScheduler.Services.Api.Abstractions.Commands
{
    public class RegisterUserCommand : ICommand
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string PasswordHash { get; set; }
        public string Email { get; set; }
    }
```

```
}
using System;
using ICommand = RemoteScheduler.Core.Cqrs.Abstractions.Contracts.ICommand;
namespace RemoteScheduler.Services.Api.Abstractions.Commands.Authorization
{
    public class SetUserRefreshTokenCommand : ICommand
    {
        public Guid UserId { get; set; }
        public Guid DeviceId { get; set; }
        public string RefreshToken { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace RemoteScheduler.Services.Api.Abstractions.Commands.Notifications.Http
{
    public class CancelHttpNotificationCommand : ICommand
    {
        public Guid NotificationId { get; set; }
        public Guid UserId { get; set; }
    }
}
using System;
using System.Collections.Generic;
using Newtonsoft.Json.Linq;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
using RemoteScheduler.Shared.Contracts.Enums;
namespace RemoteScheduler.Services.Api.Abstractions.Commands.Notifications.Http
{
    public class CreateHttpNotificationCommand : ICommand
    {
        public Guid UserId { get; set; }
        public DateTime StartAt { get; set; }
        public string CronSchedule { get; set; }
        public int RepeatsCount { get; set; }
        public TimeSpan? Interval { get; set; }
        public string Scheme { get; set; }
        public string Domain { get; set; }
        public string Path { get; set; }
        public Dictionary<string, string> Headers { get; set; }
        public Dictionary<string, string> QueryParameters { get; set; }
        public HttpActionType ActionType{ get; set; }
        public JObject Body { get; set; }
    }
}
using System;
using System.Collections.Generic;
using Newtonsoft.Json.Linq;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
using RemoteScheduler.Shared.Contracts.Enums;
namespace RemoteScheduler.Services.Api.Abstractions.Commands.Notifications.Http
{
    public class UpdateHttpNotificationCommand : ICommand
    {
        public Guid NotificationId { get; set; }
        public Guid UserId { get; set; }
        public string Scheme { get; set; }
        public string Domain { get; set; }
        public string Path { get; set; }
        public Dictionary<string, string> Headers { get; set; }
        public Dictionary<string, string> QueryParameters { get; set; }
        public HttpActionType ActionType{ get; set; }
        public JObject Body { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace RemoteScheduler.Services.Api.Abstractions.Commands.User
{
    public class ChangePasswordCommand : ICommand
    {
        public Guid UserId { get; set; }
        public string PasswordHash { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace RemoteScheduler.Services.Api.Abstractions.Commands.User
{
    public class UpdateApiKeyCommand : ICommand
    {
        public Guid UserId { get; set; }
        public string ApiKey { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace RemoteScheduler.Services.Api.Abstractions.Commands.User
{
    public class UpdateUserDetailsCommand : ICommand
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Email { get; set; }
        public Guid UserId { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace RemoteScheduler.Services.Api.Abstractions.Commands.User
{
    public class UpsertAllowedIpsCommand : ICommand
    {
        public Guid UserId { get; set; }
        public string[] AllowedIps { get; set; }
    }
}
using System;
namespace RemoteScheduler.Services.Api.Abstractions.Configurations
{
    public interface IAccessTokenConfiguration
    {
        string SecretKey { get; }
        TimeSpan Lifetime { get; }
    }
}
namespace RemoteScheduler.Services.Api.Abstractions.Configurations
{
    public interface ICorsConfiguration
    {
        ;
    }
}
namespace RemoteScheduler.Services.Api.Abstractions.Configurations
{
    public interface IDomainsConfiguration
    {
        string ApiDomain { get; }
    }
}
using RemoteScheduler.Core.Configuration;
namespace RemoteScheduler.Services.Api.Abstractions.Configurations
{
    public interface IRabbitMqConfiguration : IConfiguration
    {
        string Host { get; }
```

```
        string UserName { get; }
        string Password { get; }
        string VirtualHost { get; }
        bool AutoRecovery { get; }
    }
}
using System.Collections.Generic;
using RemoteScheduler.Core.Exceptions;
namespace RemoteScheduler.Services.Api.Abstractions.Exceptions
{
    public abstract class ApiExceptionBase :
RemoteSchedulerException
    {
        protected override IDictionary<string, string>
ExceptionMessages => new Dictionary<string, string>
        {
            ["AccessTokenDecodingException"]="Unable to decode
your access token",
            ["AccessTokenExpiredException"]="Your refresh token has
been expired. Please refresh it.",
            ["AuthorizationHeaderMissingException"]="Your IP
address is not authorized",
            ["RefreshTokenNotFoundException"]="Unable to find user
with this refresh token",
            ["UnknownAuthorizationSchemeException"]="Unknown
authorization schema",
            ["UserByApiKeyNotFoundException"]="Invalid ApiKey",
            ["UserAlreadyExistsException"]="User already exists",
            ["UserByCredentialsNotFoundException"]="Invalid
credentials",
            ["UserByRefreshTokenNotFoundException"]="User with
provided RefreshToken could not be found",
        };
        protected override IDictionary<string, string> ExceptionCodes
=> new Dictionary<string, string>
        {
            ["AccessTokenDecodingException"]="r-sched.api-001",
            ["AccessTokenExpiredException"]="r-sched.api-002",
            ["AuthorizationHeaderMissingException"]="r-sched.api-
003",
            ["DeviceIdMissingException"]="r-sched.api-004",
            ["InvalidAccessTokenException"]="r-sched.api-005",
            ["IpAddressIsNotAuthorizedException"]="r-sched.api-006",
            ["RefreshTokenNotFoundException"]="r-sched.api-007",
            ["UnknownAuthorizationSchemeException"]="r-sched.api-
008",
            ["UserByApiKeyNotFoundException"]="r-sched.api-009",
            ["UserAlreadyExistsException"]="r-sched.api-010",
            ["UserByCredentialsNotFoundException"]="r-sched.api-
011",
            ["UserByRefreshTokenNotFoundException"]="r-sched.api-
012",
        };
    }
}stem;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace RemoteScheduler.Services.Api.Abstractions.Queries
{
    public interface IUserQuery<TResponse> : IQuery<TResponse>
        where TResponse : IQueryResponse
    {
        Guid UserId { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace RemoteScheduler.Services.Api.Abstractions.Queries
{
    public class UserDetailsQuery : IQuery<UserDetailsResponse>
    {
        public Guid UserId { get; set; }
    }
    public class UserDetailsResponse : IQueryResponse
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Email { get; set; }
        public string ApiKey { get; set; }
        public string[] AllowedServerIps { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Services.Api.Abstractions.Queries.Authorization
{
    public class UserApiKeyInfoByIdQuery :
IQuery<UserApiKeyInfoByIdResponse>
    {
        public string ApiKey { get; set; }
    }
    public class UserApiKeyInfoByIdResponse : IQueryResponse
    {
        public Guid UserId { get; set; }
        public string[] AllowedIps { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Services.Api.Abstractions.Queries.Authorization
{
    public class UserByCredentialsQuery :
IQuery<UserByCredentialsResponse>
    {
        public string Email { get; set; }
        public string PasswordHash { get; set; }
    }
    public class UserByCredentialsResponse : IQueryResponse
    {
        public Guid UserId { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Services.Api.Abstractions.Queries.Authorization
{
    public class UserByRefreshTokenQuery :
IQuery<UserByRefreshTokenResponse>
    {
        public Guid DeviceId { get; set; }
        public string RefreshToken { get; set; }
    }
    public class UserByRefreshTokenResponse : IQueryResponse
    {
        public Guid UserId { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Services.Api.Abstractions.Queries.Authorization
{
    public class UserRefreshTokenByIdQuery :
IQuery<UserRefreshTokenByIdResponse>
    {
        public Guid UserId { get; set; }
        public Guid DeviceId { get; set; }
    }
    public class UserRefreshTokenByIdResponse : IQueryResponse
    {
        public string RefreshToken { get; set; }
    }
}
using System;
using System.Collections.Generic;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
using RemoteScheduler.Shared.Contracts.Enums.Jobs;
namespace
RemoteScheduler.Services.Api.Abstractions.Queries.Notifications.
Http
{
```

```
    public class HttpNotificationsQuery : IQuery<HttpNotificationsResponse>
    {
        public Guid UserId { get; set; }
    }
    public class HttpNotificationsResponse : IQueryResponse
    {
        public ICollection<HttpNotification> Notifications { get; set; }
        public class HttpNotification
        {
            public Guid Id { get; set; }
            public Guid JobId { get; set; }
            public string Scheme { get; set; }
            public string Domain { get; set; }
            public string Path { get; set; }
            public IDictionary<string, string> Headers { get; set; }
            public IDictionary<string, string> QueryParameters { get; set; }
            public JobStatus Status { get; set; }
            public string CronExpression { get; set; }
            public bool IsCron { get; set; }
            public TimeSpan? Interval { get; set; }
            public int RepeatsCount { get; set; }
            public DateTime StartAt { get; set; }
        }
    }
}
using System;
namespace RemoteScheduler.Services.Api.Abstractions.Services.Authorization
{
    public interface IAccessTokenPayload
    {
        Guid UserId { get; set; }
        DateTime ExpirationDate { get; set; }
        long Lifetime { get; set; }
    }
}
namespace RemoteScheduler.Services.Api.Abstractions.Services.Authorization
{
    public interface IAccessTokenService
    {
        string CreateToken<TPayload>(TPayload payload, string customSecret) where TPayload : IAccessTokenPayload;
        TPayload GetPayload<TPayload>(string token) where TPayload : IAccessTokenPayload;
        TPayload ValidateToken<TPayload>(string token, string customSecret) where TPayload : IAccessTokenPayload;
    }
}
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
namespace RemoteScheduler.Services.Api.Abstractions.Services.Authorization
{
    public interface IAuthorizationSchemeProcessor
    {
        Task<TPayload> Authorize<TPayload>(HttpRequest httpRequest, string token) where TPayload : IAccessTokenPayload, new();
    }
}
namespace RemoteScheduler.Services.Api.Abstractions.Services.Hashing
{
    public interface IRandomStringGenerationService
    {
        string Generate(int bytesLength);
    }
}
namespace RemoteScheduler.Services.Api.Abstractions.Services.Hashing
{
    public interface ISha512HashingService
    {
        string GenerateHash(string source);
    }
}
using Autofac;
using RemoteScheduler.Services.Api.Database.Context;
namespace RemoteScheduler.Services.Api.Database
{
    public class ApiDatabaseModule : Module
    {
        protected override void Load(ContainerBuilder builder)
        {
            builder.RegisterType<RemoteSchedulerDbContext>()
                .AsImplementedInterfaces();
        }
    }
}
using System.Collections.Generic;
using System.Data;
namespace RemoteScheduler.Services.Api.Database.Context
{
    public interface IRemoteSchedulerDbContext
    {
        DbSet<User> Users { get; }
        DbSet<UserRefreshToken> UserRefreshTokens { get; }
        DbSet<Job> Jobs { get; }
        DbSet<JobExecution> JobExecutions { get; }
        DbSet<HttpNotification> HttpNotifications { get; }
        DbSet<HttpNotificationHeader> HttpNotificationHeaders { get; }
        DbSet<HttpNotificationQueryParameter> HttpNotificationQueryParameters { get; }
        DbSet<UserEmailSettings> UserEmailSettings { get; }
        DbSet<EmailReceiver> EmailReceivers { get; }
        DbSet<EmailNotificationReceiver> EmailNotificationReceivers { get; }
        DbSet<EmailNotification> EmailNotifications { get; }
        #region Context methods
        IDbConnection Connection { get; }
        IEnumerable<TEntity> AddRange<TEntity>(IEnumerable<TEntity> entities) where TEntity : class;
        EntityEntry<TEntity> Entry<TEntity>(TEntity entity) where TEntity : class;
        IEnumerable<TEntity> RemoveRange<TEntity>(IEnumerable<TEntity> entities) where TEntity : class;
        Task<int> SaveChangesAsync(CancellationToken cancellationToken = new CancellationToken());
        IQueryable<TEntity> RawQuery<TEntity>(string query, params object[] arguments) where TEntity : class;
        #endregion
    }
}
using System;
using System.Collections.Generic;
using RemoteScheduler.Services.Api.Database.Entities.Users;
using RemoteScheduler.Services.Api.Database.Extensions;
namespace RemoteScheduler.Services.Api.Database.Context
{
    public class RemoteSchedulerDbContext : DbContext, IRemoteSchedulerDbContext
    {
        private readonly string _connectionString;
        public DbSet<User> Users { get; set; }
        public DbSet<UserRefreshToken> UserRefreshTokens { get; set; }
        public DbSet<Job> Jobs { get; set; }
        public DbSet<JobExecution> JobExecutions { get; set; }
        public DbSet<HttpNotification> HttpNotifications { get; set; }
        public DbSet<HttpNotificationHeader> HttpNotificationHeaders { get; set; }
        public DbSet<HttpNotificationQueryParameter> HttpNotificationQueryParameters { get; set; }
        public DbSet<UserEmailSettings> UserEmailSettings { get; set; }
        public DbSet<EmailReceiver> EmailReceivers { get; set; }
```

```
        public DbSet<EmailNotificationReceiver> EmailNotificationReceivers { get; set; }
        public DbSet<EmailNotification> EmailNotifications { get; set; }
        public RemoteSchedulerDbContext(IConfigurationRoot configurationRoot)
        {
            _connectionString = configurationRoot.GetConnectionString(nameof(RemoteSchedulerDbContext));
        }
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseNpgsql(_connectionString, builder =>
            {
                var assemblyName = Assembly.GetExecutingAssembly().FullName;
                builder.MigrationsAssembly(assemblyName);
            });
        }
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.HasPostgresExtension("uuid-ossp");
            modelBuilder.ToSnakeCase();

            modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());
            AddDefaultSqlValues(modelBuilder);
        }
        #region ContextMethods
        public IDbConnection Connection => Database.GetDbConnection();
        public IEnumerable<TEntity> AddRange<TEntity>(IEnumerable<TEntity> entities) where TEntity : class
        {
            Set<TEntity>().AddRange(entities);
            return Set<TEntity>();
        }
        public IEnumerable<TEntity> RemoveRange<TEntity>(IEnumerable<TEntity> entities) where TEntity : class
        {
            Set<TEntity>().RemoveRange(entities);
            return Set<TEntity>();
        }
        public IQueryable<TEntity> RawQuery<TEntity>(string query, params object[] arguments) where TEntity : class
        {
            return Set<TEntity>().FromSql(query, arguments);
        }
        private static void AddDefaultSqlValues(ModelBuilder modelBuilder)
        {
            var mutableEntityTypes = modelBuilder.Model.GetEntityTypes()
                .Where(type => type.ClrType.IsAssignableTo<EntityBase>()).ToList();
            foreach (var mType in mutableEntityTypes)
            {
                modelBuilder.Entity(mType.ClrType)
                    .Property("Id")
                    .HasDefaultValueSql("uuid_generate_v4()");
            }
        }
        public override Task<int> SaveChangesAsync(CancellationToken cancellationToken = new CancellationToken())
        {
            foreach (var dbEntityEntry in ChangeTracker.Entries())
            {
                if (!(dbEntityEntry.Entity is EntityBase entityBase))
                {
                    continue;
                }
                if (dbEntityEntry.State == EntityState.Added)
                {
                    entityBase.CreationDate = DateTime.UtcNow;
                    entityBase.UpdateDate = DateTime.UtcNow;
                }
                if (dbEntityEntry.State == EntityState.Modified)
                {
                    entityBase.UpdateDate = DateTime.UtcNow;
                }
            }
            return base.SaveChangesAsync(cancellationToken);
        }
        #endregion
    }
}
using System;
using System.ComponentModel.DataAnnotations;
namespace RemoteScheduler.Services.Api.Database.Entities.Base
{
    public class EntityBase : IEntity
    {
        [Key]
        public virtual Guid Id { get; set; }
        public DateTime CreationDate { get; set; }
        public DateTime UpdateDate { get; set; }
    }
}
using System;
namespace RemoteScheduler.Services.Api.Database.Entities.Base
{
    public interface IEntity
    {
        DateTime CreationDate { get; set; }
        DateTime UpdateDate { get; set; }
    }
}
using System;
namespace RemoteScheduler.Services.Api.Database.Entities.EmailNotifications
{
    public class EmailNotification : EntityBase
    {
        [ForeignKey(nameof(Job))]
        public Guid JobId { get; set; }
        public Job Job { get; set; }
        [ForeignKey(nameof(UserEmailSettings))]
        public Guid? UserEmailSettingsId { get; set; }
        public UserEmailSettings UserEmailSettings { get; set; }
        public ICollection<EmailNotificationReceiver> Receivers { get; set; }
    }
}
using RemoteScheduler.Services.Api.Database.Entities.Base;
namespace RemoteScheduler.Services.Api.Database.Entities.EmailNotifications
{
    public class EmailNotificationReceiver : EntityBase
    {
        [ForeignKey(nameof(EmailNotification))]
        public Guid NotificationId { get; set; }
        public EmailNotification EmailNotification { get; set; }
        [ForeignKey(nameof(EmailReceiver))]
        public Guid ReceiverId { get; set; }
        public EmailReceiver EmailReceiver { get; set; }
    }
}
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
namespace RemoteScheduler.Services.Api.Database.Entities.EmailNotifications
{
    public class EmailReceiver : EntityBase
    {
```

```
        [ForeignKey(nameof(User))]
        public Guid UserId { get; set; }
        public User User { get; set; }
        public string Email { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        [Column(TypeName = "jsonb")]
        public JObject Data { get; set; }
        public ICollection<EmailNotification> Notifications { get; set;
}
    }
}
using System;
using System.Collections.Generic.Notifications
{
    public class EmailTemplate : EntityBase
    {
        [ForeignKey(nameof(User))]
        public Guid UserId { get; set; }
        public User User { get; set; }
        [StringLength(128)]
        public string Key { get; set; }
        [StringLength(16384)]
        public string Body { get; set; }
        public ICollection<EmailNotification> Notifications { get; set;
}
    }
}
using System;
using System.ComponentModel.DataAnnotations.Schema;
namespace
RemoteScheduler.Services.Api.Database.Entities.EmailNotification
s
{
    public class UserEmailSettings : EntityBase
    {
        [ForeignKey(nameof(User))]
        public Guid UserId { get; set; }
        public User User { get; set; }
        public string Name { get; set; }
        public string SenderEmail { get; set; }
        public string Login { get; set; }
        public string Password { get; set; }
        public string Host { get; set; }
        public int Port { get; set; }
        public bool UseSsl { get; set; }
    }
}
using System;
using System.Collections.Generic;
using RemoteScheduler.Shared.Contracts.Enums;
namespace
RemoteScheduler.Services.Api.Database.Entities.HttpNotifications
{
    public class HttpNotification : EntityBase
    {
        [ForeignKey(nameof(Job))]
        public Guid JobId { get; set; }
        public Job Job { get; set; }
        public string Schema { get; set; }
        public string Domain { get; set; }
        public string Path { get; set; }
        public HttpActionType ActionType { get; set; }
        [Column(TypeName = "jsonb")]
        public JObject Body { get; set; }
        public ICollection<HttpNotificationHeader> Headers { get;
set; }
        public ICollection<HttpNotificationQueryParameter>
QueryParameters { get; set; }
    }
}
using System;
using System.ComponentModel.DataAnnotations.Schema;
using RemoteScheduler.Services.Api.Database.Entities.Base;
namespace
RemoteScheduler.Services.Api.Database.Entities.HttpNotifications
{
    public class HttpNotificationHeader : EntityBase
    {
        [ForeignKey(nameof(Notification))]
        public Guid NotificationId { get; set; }
        public HttpNotification Notification { get; set; }
        public string Key { get; set; }
        public string Value { get; set; }
    }
}
using System;
using System.ComponentModel.DataAnnotations.Schema;
using RemoteScheduler.Services.Api.Database.Entities.Base;
namespace
RemoteScheduler.Services.Api.Database.Entities.HttpNotifications
{
    public class HttpNotificationQueryParameter : EntityBase
    {
        [ForeignKey(nameof(Notification))]
        public Guid NotificationId { get; set; }
        public HttpNotification Notification { get; set; }
        public string Key { get; set; }
        public string Value { get; set; }
    }
}
using System;
using System.Collections.Generic;
namespace RemoteScheduler.Services.Api.Database.Entities.Jobs
{
    public class Job : EntityBase
    {
        [ForeignKey(nameof(Owner))]
        public Guid OwnerId { get; set; }
        public User Owner { get; set; }
        public JobType Type { get; set; }
        public JobStatus Status { get; set; }
        public string CommandType { get; set; }
        public DateTime StartAt { get; set; }
        public bool IsCron { get; set; }
        public string CronSchedule { get; set; }
        public int RepeatsCount { get; set; }
        public TimeSpan? Interval { get; set; }
        public ICollection<JobExecution> JobExecutions { get; set; }
    }
}
using System;
using System.ComponentModel.DataAnnotations.Schema;
namespace RemoteScheduler.Services.Api.Database.Entities.Jobs
{
    public class JobExecution : EntityBase
    {
        [ForeignKey(nameof(Job))]
        public Guid JobId { get; set; }
        public Job Job { get; set; }
        public JobExecutionStatus ExecutionStatus { get; set; }
    }
}
using System.Collections.Generic;
namespace RemoteScheduler.Services.Api.Database.Entities.Users
{
    public class User : EntityBase
    {
        [StringLength(200)]
        public string FirstName { get; set; }
        [StringLength(200)]
        public string LastName { get; set; }
        [StringLength(200)]
        public string Email { get; set; }
        [StringLength(300)]
        public string PasswordHash { get; set; }
        [StringLength(196)]
        public string ApiKey { get; set; }
        [Column(TypeName = "varchar(32)[]")]
        public string[] AllowedServerIps { get; set; }
        public ICollection<UserRefreshToken> RefreshTokens { get;
set; }
    }
}
```

```
using System;
using System.ComponentModel.DataAnnotations.Schema;
using RemoteScheduler.Services.Api.Database.Entities.Base;
namespace RemoteScheduler.Services.Api.Database.Entities.Users
{
    public class UserRefreshToken : EntityBase
    {
        [ForeignKey(nameof(User))]
        public Guid UserId { get; set; }
        public User User { get; set; }
        public Guid DeviceId { get; set; }
        public string RefreshToken { get; set; }
    }
}
using Microsoft.EntityFrameworkCore;
namespace
RemoteScheduler.Services.Api.Database.EntityConfigurations.Notifications
{
    public class EmailReceiverConfiguration :
IEntityTypeConfiguration<EmailReceiver>
    {
        public void Configure(EntityTypeBuilder<EmailReceiver>
builder)
        {
            builder.Property(notification => notification.Data)
                .HasConversion(
                    o => o != null ? o.ToString() : null,
                    s => s != null ? JObject.Parse(s) : null);
        }
    }
}
using Microsoft.EntityFrameworkCore;
namespace
RemoteScheduler.Services.Api.Database.EntityConfigurations.Notifications
{
    public class EmailTemplateConfiguration :
IEntityTypeConfiguration<EmailTemplate>
    {
        public void Configure(EntityTypeBuilder<EmailTemplate>
builder)
        {
            builder.HasIndex(template => new
            {
                template.UserId,
                template.Key
            }).IsUnique();
        }
    }
}
using Microsoft.EntityFrameworkCore;
namespace
RemoteScheduler.Services.Api.Database.EntityConfigurations.Notifications
{
    public class HttpNotificationEntityConfiguration :
IEntityTypeConfiguration<HttpNotification>
    {
        public void Configure(EntityTypeBuilder<HttpNotification>
builder)
        {
            builder.Property(notification => notification.Body)
                .HasConversion(
                    o => o != null ? o.ToString() : null,
                    s => s != null ? JObject.Parse(s) : null);
        }
    }
}
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using RemoteScheduler.Services.Api.Database.Entities.Users;
namespace
RemoteScheduler.Services.Api.Database.EntityConfigurations.Users
{
    public class UserConfiguration :
IEntityTypeConfiguration<User>
    {
        public void Configure(EntityTypeBuilder<User> builder)
        {
            builder.HasIndex(user => user.Email)
                .IsUnique()
                .HasName("ux_users_email");
        }
    }
}
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using RemoteScheduler.Services.Api.Database.Entities.Users;
namespace
RemoteScheduler.Services.Api.Database.EntityConfigurations.Users
{
    public class UserRefreshTokenConfiguration :
IEntityTypeConfiguration<UserRefreshToken>
    {
        public void
Configure(EntityTypeBuilder<UserRefreshToken> builder)
        {
            builder.HasIndex(token => new
                {
                    token.RefreshToken,
                    token.DeviceId
                })
                .IsUnique(true)

.HasName("ux_user_refresh_tokens_refresh_token_device_id");
        }
    }
}
using System.Text.RegularExpressions;
using Microsoft.EntityFrameworkCore;
namespace RemoteScheduler.Services.Api.Database.Extensions
{
    public static class NamingExtensions
    {
        public static string ToSnakeCase(this string input)
        {
            if (string.IsNullOrEmpty(input))
            {
                return input;
            }
            var noLeadingUndescore = Regex.Replace(input, @"^_",
"");
            return Regex.Replace(noLeadingUndescore, @"([a-z0-
9])([A-Z])", "$1_$2").ToLower();
        }
        public static void ToSnakeCase(this ModelBuilder
modelBuilder)
        {
            foreach (var entity in
modelBuilder.Model.GetEntityTypes())
            {
                entity.Relational().TableName =
entity.Relational().TableName.ToSnakeCase();
                foreach (var property in entity.GetProperties())
                {
                    property.Relational().ColumnName =
property.Name.ToSnakeCase();
                }
                foreach (var key in entity.GetKeys())
                {
                    key.Relational().Name =
key.Relational().Name.ToSnakeCase();
                }
                foreach (var key in entity.GetForeignKeys())
                {
                    key.Relational().Name =
key.Relational().Name.ToSnakeCase();
                }
                foreach (var index in entity.GetIndexes())
                {
```

```
                index.Relational().Name =
index.Relational().Name.ToSnakeCase();
            }
        }
    }
  }
}
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
using
RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Commands;
using RemoteScheduler.Services.Api.Database.Context;
namespace RemoteScheduler.Services.Api.Database.Handlers
{
  public abstract class
ContextCommandHandlerBase<TCommand> :
CommandHandlerBase<TCommand> where TCommand :
ICommand
  {
    protected IRemoteSchedulerDbContext DbContext { get; }
    protected
ContextCommandHandlerBase(IRemoteSchedulerDbContext
dbContext)
    {
      DbContext = dbContext;
    }
  }
}
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
using RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Queries;
using RemoteScheduler.Services.Api.Database.Context;
namespace RemoteScheduler.Services.Api.Database.Handlers
{
  public abstract class ContextQueryHandlerBase<TQuery,
TResult> : QueryHandlerBase<TQuery, TResult>
    where TQuery : IQuery<TResult>
    where TResult : IQueryResponse
  {
    protected IRemoteSchedulerDbContext DbContext { get; }
    protected
ContextQueryHandlerBase(IRemoteSchedulerDbContext
dbContext)
    {
      DbContext = dbContext;
    }
  }
}

using Microsoft.EntityFrameworkCore.Migrations;
namespace RemoteScheduler.Services.Api.Database.Migrations
{
  public partial class AddedJobCompletionTrigger : Migration
  {
    protected override void Up(MigrationBuilder
migrationBuilder)
    {
      migrationBuilder.Sql(@"
        CREATE OR REPLACE FUNCTION
set_job_status_done()
          RETURNS trigger
        AS
        $$
        DECLARE
          _executed_count INT;
          DECLARE _expected_count INT;
          DECLARE _should_check BOOLEAN;
        BEGIN
          IF NEW.job_id IS NULL THEN
            RETURN NEW;
          end if;
          SELECT COUNT(id) INTO _executed_count FROM
job_executions WHERE job_id = NEW.job_id;
          SELECT repeats_count, NOT is_cron INTO
_expected_count, _should_check FROM jobs WHERE id =
NEW.job_id;
          IF (NOT _should_check) THEN
            RETURN NEW;
          end if;
          IF (_executed_count >= _expected_count) THEN
            UPDATE jobs SET status = 2 WHERE id =
NEW.job_id;
          end if;
          RETURN NEW;
        end;
        $$
          LANGUAGE 'plpgsql';
        CREATE TRIGGER job_executed
          AFTER INSERT
          ON job_executions
          FOR EACH ROW
          EXECUTE PROCEDURE set_job_status_done();");
    }
    protected override void Down(MigrationBuilder
migrationBuilder)
    {
      migrationBuilder.Sql(@"DROP TRIGGER job_executed
ON job_executions;");
      migrationBuilder.Sql(@"DROP FUNCTION
set_job_status_done;");
    }
  }
}

using System;
namespace RemoteScheduler.Services.Api.Database.DesignTime
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Do nothing here.");
    }
  }
}
using Autofac;
using
RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Commands;
using RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Events;
using RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Queries;
using RemoteScheduler.Services.Api.Implementation.Services;
using
RemoteScheduler.Services.Api.Implementation.Services.Hashing;
namespace RemoteScheduler.Services.Api.Implementation
{
  public class ApiImplementationModule : Module
  {
    protected override void Load(ContainerBuilder builder)
    {
      builder.RegisterAssemblyTypes(ThisAssembly)
        .AsClosedTypesOf(typeof(ICommandHandler<>))
        .AsImplementedInterfaces();
      builder.RegisterAssemblyTypes(ThisAssembly)
        .AsClosedTypesOf(typeof(IEventSubscriber<>))
        .AsImplementedInterfaces();
      builder.RegisterAssemblyTypes(ThisAssembly)
        .AsClosedTypesOf(typeof(IQueryHandler<,>))
        .AsImplementedInterfaces();

builder.RegisterType<Sha512HashingService>().AsImplementedIn
terfaces();

builder.RegisterType<RandomStringGenerationService>().AsImple
mentedInterfaces();

builder.RegisterType<AccessTokenService>().AsImplementedInter
faces();
    }
  }
}
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using RemoteScheduler.Services.Api.Database.Handlers;
namespace
RemoteScheduler.Services.Api.Implementation.Handlers.Comman
ds
```

```
{
  public class RegisterUserCommandHandler :
ContextCommandHandlerBase<RegisterUserCommand>
  {
    private readonly IRandomStringGenerationService
_randomStringGenerationService;
    public
RegisterUserCommandHandler(IRemoteSchedulerDbContext
dbContext,
      IRandomStringGenerationService
randomStringGenerationService) : base(dbContext)
    {
      _randomStringGenerationService =
randomStringGenerationService;
    }
    public override async Task Handle(RegisterUserCommand
command)
    {
      var emailTaken = await DbContext.Users.AnyAsync(user
=> user.Email == command.Email);
      if (emailTaken)
      {
        throw new UserAlreadyExistsException();
      }
      var userToAdd = new Database.Entities.Users.User
      {
        Email = command.Email,
        FirstName = command.FirstName,
        LastName = command.LastName,
        PasswordHash = command.PasswordHash,
        ApiKey =
_randomStringGenerationService.Generate(96),
        AllowedServerIps = new string[0]
      };
      DbContext.Users.Add(userToAdd);
      await DbContext.SaveChangesAsync();
    }
  }
}
using System.Linq;
using System.Threading.Tasks;
namespace
RemoteScheduler.Services.Api.Implementation.Handlers.Comman
ds.Authorization
{
  public class SetUserRefreshTokenCommandHandler :
ContextCommandHandlerBase<SetUserRefreshTokenCommand>
  {
    public
SetUserRefreshTokenCommandHandler(IRemoteSchedulerDbCont
ext dbContext) : base(dbContext)
    {
    }
    public override async Task
Handle(SetUserRefreshTokenCommand command)
    {
      var userToken = await
DbContext.UserRefreshTokens.FirstOrDefaultAsync(token =>
        token.UserId == command.UserId && token.DeviceId
== command.DeviceId);
      if (userToken == null)
      {
        var userRefreshToken = new UserRefreshToken
        {
          DeviceId = command.DeviceId,
          RefreshToken = command.RefreshToken,
          UserId = command.UserId
        };
        DbContext.UserRefreshTokens.Add(userRefreshToken);
      }
      else
      {
        userToken.RefreshToken = command.RefreshToken;
      }
      await DbContext.SaveChangesAsync();
    }
  }
}
using System.Threading.Tasks;
using RemoteScheduler.Shared.Contracts.Events.Notifications;
namespace
RemoteScheduler.Services.Api.Implementation.Handlers.Comman
ds.Notifications.Http
{
  public class CancelHttpNotificationCommandHandler :
ContextCommandHandlerBase<CancelHttpNotificationCommand>
  {
    private readonly IEventBus _eventBus;
    public
CancelHttpNotificationCommandHandler(IRemoteSchedulerDbCo
ntext dbContext, IEventBus eventBus) :
      base(dbContext)
    {
      _eventBus = eventBus;
    }
    public override async Task
Handle(CancelHttpNotificationCommand command)
    {
      var httpNotification = await DbContext.HttpNotifications
        .Include(notification => notification.Job)
        .FirstOrDefaultAsync(notification =>
          notification.Id == command.NotificationId &&
          notification.Job.OwnerId == command.UserId);
      if (httpNotification == null)
      {
        throw new
EntityNotFoundException<CancelHttpNotificationCommand,
HttpNotification>(command);
      }
      httpNotification.Job.Status = JobStatus.Canceled;
      await DbContext.SaveChangesAsync();
      await _eventBus.Raise(new HttpNotificationCanceledEvent
      {
        JobId = httpNotification.JobId,
        NotificationId = httpNotification.Id
      });
    }
  }
}
using System.Collections.Generic;
using RemoteScheduler.Shared.Contracts.Events.Notifications;
namespace
RemoteScheduler.Services.Api.Implementation.Handlers.Comman
ds.Notifications.Http
{
  public class CreateHttpNotificationCommandHandler :
ContextCommandHandlerBase<CreateHttpNotificationCommand>
  {
    private readonly IEventBus _eventBus;
    public
CreateHttpNotificationCommandHandler(IRemoteSchedulerDbCon
text dbContext, IEventBus eventBus) :
      base(dbContext)
    {
      _eventBus = eventBus;
    }
    public override async Task
Handle(CreateHttpNotificationCommand command)
    {
      var job = new Job
      {
        OwnerId = command.UserId,
        Status = JobStatus.Created,
        CommandType =
typeof(SendHttpNotificationCommand).Name,
        RepeatsCount = command.RepeatsCount,
        Interval = command.Interval,
        Type = JobType.HttpNotification,
        CronSchedule = command.CronSchedule,
        IsCron = command.CronSchedule != null,
        StartAt = command.StartAt
      };
      var notification = new HttpNotification
      {
```

```
            Job = job,
            Body = command.Body,
            Domain = command.Domain,
            Path = command.Path,
            Schema = command.Scheme,
            ActionType = command.ActionType,
            Headers = command.Headers?.Select(pair => new
HttpNotificationHeader
            {
               Key = pair.Key,
               Value = pair.Value
            }).ToList(),
            QueryParameters =
command.QueryParameters?.Select(pair => new
HttpNotificationQueryParameter
            {
               Key = pair.Key,
               Value = pair.Value
            }).ToList()
         };
         DbContext.Jobs.Add(job);
         DbContext.HttpNotifications.Add(notification);
         await DbContext.SaveChangesAsync();
         await _eventBus.Raise(new HttpNotificationCreatedEvent
         {
            Id = notification.Id
         });
      }
   }
}
using System.Linq;
using System.Threading.Tasks;
using RemoteScheduler.Shared.Contracts.Events.Notifications;
namespace
RemoteScheduler.Services.Api.Implementation.Handlers.Comman
ds.Notifications.Http
{
   public class UpdateHttpNotificationCommandHandler :
ContextCommandHandlerBase<UpdateHttpNotificationCommand
>
   {
      private readonly IEventBus _eventBus;
      public
UpdateHttpNotificationCommandHandler(IRemoteSchedulerDbCo
ntext dbContext, IEventBus eventBus) :
         base(dbContext)
      {
         _eventBus = eventBus;
      }
      public override async Task
Handle(UpdateHttpNotificationCommand command)
      {
         var httpNotification = await DbContext.HttpNotifications
            .Include(notification => notification.Headers)
            .Include(notification => notification.QueryParameters)
            .FirstOrDefaultAsync(notification =>
               notification.Id == command.NotificationId
               && notification.Job.OwnerId == command.UserId
               && notification.Job.Status == JobStatus.Scheduled);
         if (httpNotification == null)
         {
            throw new
EntityNotFoundException<UpdateHttpNotificationCommand,
HttpNotification>(command);
         }
         httpNotification.Body = command.Body;
         httpNotification.Domain = command.Domain;
         httpNotification.Path = command.Path;
         httpNotification.ActionType = command.ActionType;
         httpNotification.Schema = command.Scheme;
         ProcessHeaders(command, httpNotification);
         ProcessQueryParameters(command, httpNotification);
         await DbContext.SaveChangesAsync();
         await _eventBus.Raise(new HttpNotificationUpdatedEvent
         {
            NotificationId = command.NotificationId
         });
      }
      private void
ProcessQueryParameters(UpdateHttpNotificationCommand
command, HttpNotification httpNotification)
      {
         if (command.QueryParameters == null)
         {

DbContext.RemoveRange(httpNotification.QueryParameters);
            httpNotification.QueryParameters = null;
         }
         else
         {
            if (httpNotification.QueryParameters == null)
            {
               httpNotification.QueryParameters =
command.QueryParameters.Select(pair =>
                  new HttpNotificationQueryParameter
                  {
                     Key = pair.Key,
                     Value = pair.Value
                  }).ToList();
            }
            else
            {

DbContext.RemoveRange(httpNotification.QueryParameters);
               httpNotification.QueryParameters =
command.QueryParameters.Select(pair =>
                  new HttpNotificationQueryParameter
                  {
                     Key = pair.Key,
                     Value = pair.Value
                  }).ToList();
            }
         }
      }
      private void
ProcessHeaders(UpdateHttpNotificationCommand command,
HttpNotification httpNotification)
      {
         if (command.Headers == null)
         {
            DbContext.RemoveRange(httpNotification.Headers);
            httpNotification.Headers = null;
         }
         else
         {
            if (httpNotification.Headers == null)
            {
               httpNotification.Headers =
command.Headers.Select(pair =>
                  new HttpNotificationHeader
                  {
                     Key = pair.Key,
                     Value = pair.Value
                  }).ToList();
            }
            else
            {
               DbContext.RemoveRange(httpNotification.Headers);
               httpNotification.Headers =
command.Headers.Select(pair =>
                  new HttpNotificationHeader
                  {
                     Key = pair.Key,
                     Value = pair.Value
                  }).ToList();
            }
         }
      }
   }
}
using System.Threading.Tasks;
namespace
RemoteScheduler.Services.Api.Implementation.Handlers.Comman
ds.User
```

```
{
  public class ChangePasswordCommandHandler : ContextCommandHandlerBase<ChangePasswordCommand>
  {
    public ChangePasswordCommandHandler(IRemoteSchedulerDbContext dbContext) : base(dbContext)
    {
    }
    public override async Task Handle(ChangePasswordCommand command)
    {
      var userToUpdate = await DbContext.Users.FirstOrDefaultAsync(user => user.Id == command.UserId);
      if (userToUpdate == null)
      {
        throw new EntityNotFoundException<ChangePasswordCommand, Database.Entities.Users.User>(command);
      }
      userToUpdate.PasswordHash = command.PasswordHash;
      await DbContext.SaveChangesAsync();
    }
  }
}
using System.Threading.Tasks;
namespace RemoteScheduler.Services.Api.Implementation.Handlers.Commands.User
{
  public class UpdateApiKeyCommandHandler : ContextCommandHandlerBase<UpdateApiKeyCommand>
  {
    public UpdateApiKeyCommandHandler(IRemoteSchedulerDbContext dbContext) : base(dbContext)
    {
    }
    public override async Task Handle(UpdateApiKeyCommand command)
    {
      var userToUpdate = await DbContext.Users.FirstOrDefaultAsync(user => user.Id == command.UserId);
      if (userToUpdate == null)
      {
        throw new EntityNotFoundException<UpdateApiKeyCommand, Database.Entities.Users.User>(command);
      }
      userToUpdate.ApiKey = command.ApiKey;
      await DbContext.SaveChangesAsync();
    }
  }
}
using System.Linq;
using System.Threading.Tasks;
namespace RemoteScheduler.Services.Api.Implementation.Handlers.Commands.User
{
  public class UpdateUserDetailsCommandHandler : ContextCommandHandlerBase<UpdateUserDetailsCommand>
  {
    public UpdateUserDetailsCommandHandler(IRemoteSchedulerDbContext dbContext) : base(dbContext)
    {
    }
    public override async Task Handle(UpdateUserDetailsCommand command)
    {
      var users = await DbContext.Users.Where(user => user.Id == command.UserId || user.Email == command.Email)
        .ToListAsync();
      if (!users.Any())
      {
        throw new EntityNotFoundException<UpdateUserDetailsCommand, Database.Entities.Users.User>(command);
      }
      var userWithSameEmail = users.FirstOrDefault(user => user.Email == command.Email && user.Id != command.UserId);
      if (null != userWithSameEmail)
      {
        throw new UserAlreadyExistsException();
      }
      var userToUpdate = users.FirstOrDefault(user => user.Id == command.UserId);
      if (userToUpdate == null)
      {
        throw new EntityNotFoundException<UpdateUserDetailsCommand, Database.Entities.Users.User>(command);
      }
      userToUpdate.Email = command.Email;
      userToUpdate.FirstName = command.FirstName;
      userToUpdate.LastName = command.LastName;
      await DbContext.SaveChangesAsync();
    }
  }
}
using System.Threading.Tasks;
namespace RemoteScheduler.Services.Api.Implementation.Handlers.Commands.User
{
  public class UpsertAllowedIpsCommandHandler : ContextCommandHandlerBase<UpsertAllowedIpsCommand>
  {
    public UpsertAllowedIpsCommandHandler(IRemoteSchedulerDbContext dbContext) : base(dbContext)
    {
    }
    public override async Task Handle(UpsertAllowedIpsCommand command)
    {
      var userToUpdate = await DbContext.Users.FirstOrDefaultAsync(user => user.Id == command.UserId);
      if (userToUpdate == null)
      {
        throw new EntityNotFoundException<UpsertAllowedIpsCommand, Database.Entities.Users.User>(command);
      }
      userToUpdate.AllowedServerIps = command.AllowedIps;
      await DbContext.SaveChangesAsync();
    }
  }
}
using RemoteScheduler.Services.Api.Database.Context;
namespace RemoteScheduler.Services.Api.Implementation.Handlers.Events.Jobs
{
  public class SetJobStatusCanceledEventSubscriber : SetJobStatusEventSubscriberBase<JobCanceledEvent>
  {
    public SetJobStatusCanceledEventSubscriber(IRemoteSchedulerDbContext dbContext) : base(dbContext)
    {
    }
    protected override JobStatus StatusToSet => JobStatus.Canceled;
  }
}
using System.Threading.Tasks;
```

```
namespace RemoteScheduler.Services.Api.Implementation.Handlers.Events.Jobs
{
    public abstract class SetJobStatusEventSubscriberBase<TEvent> : EventSubscriberBase<TEvent> where TEvent : IJobEvent
    {
        private readonly IRemoteSchedulerDbContext _dbContext;
        protected SetJobStatusEventSubscriberBase(IRemoteSchedulerDbContext dbContext)
        {
            _dbContext = dbContext;
        }
        protected abstract JobStatus StatusToSet { get; }
        public override async Task Handle(TEvent @event)
        {
            var jobToSetStatus = await _dbContext.Jobs.FirstOrDefaultAsync(job => job.Id == @event.JobId);
            if (jobToSetStatus == null)
            {
                throw new EntityNotFoundException<TEvent, Job>(@event);
            }
            jobToSetStatus.Status = StatusToSet;
            await _dbContext.SaveChangesAsync();
        }
    }
}
using RemoteScheduler.Services.Api.Database.Context;
namespace RemoteScheduler.Services.Api.Implementation.Handlers.Events.Jobs
{
    public class SetJobStatusScheduledEventSubscriber : SetJobStatusEventSubscriberBase<JobScheduledEvent>
    {
        public SetJobStatusScheduledEventSubscriber(IRemoteSchedulerDbContext dbContext) : base(dbContext)
        {
        }
        protected override JobStatus StatusToSet => JobStatus.Scheduled;
    }
}
using System.Threading.Tasks;
namespace RemoteScheduler.Services.Api.Implementation.Handlers.Events.Jobs.Executions
{
    public class CreateJobExecutionEventSubscriber : EventSubscriberBase<JobExecutionStartedEvent>
    {
        private readonly IRemoteSchedulerDbContext _dbContext;
        public CreateJobExecutionEventSubscriber(IRemoteSchedulerDbContext dbContext)
        {
            _dbContext = dbContext;
        }
        public override async Task Handle(JobExecutionStartedEvent @event)
        {
            var jobExecution = new JobExecution
            {
                Id = @event.ExecutionId,
                JobId = @event.JobId,
                ExecutionStatus = JobExecutionStatus.Started
            };
            _dbContext.JobExecutions.Add(jobExecution);
            await _dbContext.SaveChangesAsync();
        }
    }
}
using System.Threading.Tasks;
namespace RemoteScheduler.Services.Api.Implementation.Handlers.Events.Jobs.Executions
{
    public class SetJobJobExecutionStatusFailedEventSubscriber : EventSubscriberBase<JobExecutionFailedEvent>
    {
        private readonly IRemoteSchedulerDbContext _dbContext;
        public SetJobJobExecutionStatusFailedEventSubscriber(IRemoteSchedulerDbContext dbContext)
        {
            _dbContext = dbContext;
        }
        public override async Task Handle(JobExecutionFailedEvent @event)
        {
            var jobExecution = await _dbContext.JobExecutions.FirstOrDefaultAsync(execution => execution.Id == @event.ExecutionId);
            if (jobExecution == null)
            {
                throw new EntityNotFoundException<JobExecutionFailedEvent, JobExecution>(@event);
            }
            jobExecution.ExecutionStatus = JobExecutionStatus.Failed;
            await _dbContext.SaveChangesAsync();
        }
    }
}
using System.Threading.Tasks;
namespace RemoteScheduler.Services.Api.Implementation.Handlers.Events.Jobs.Executions
{
    public class SetJobJobExecutionStatusSucceedEventSubscriber : EventSubscriberBase<JobExecutionSucceedEvent>
    {
        private readonly IRemoteSchedulerDbContext _dbContext;
        public SetJobJobExecutionStatusSucceedEventSubscriber(IRemoteSchedulerDbContext dbContext)
        {
            _dbContext = dbContext;
        }
        public override async Task Handle(JobExecutionSucceedEvent @event)
        {
            var jobExecution = await _dbContext.JobExecutions.FirstOrDefaultAsync(execution => execution.Id == @event.ExecutionId);
            if (jobExecution == null)
            {
                throw new EntityNotFoundException<JobExecutionSucceedEvent, JobExecution>(@event);
            }
            jobExecution.ExecutionStatus = JobExecutionStatus.Succeed;
            await _dbContext.SaveChangesAsync();
        }
    }
}
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
namespace RemoteScheduler.Services.Api.Implementation.Handlers.Events.Notifications.Http
{
    public class ScheduleHttpNotificationEventSubscriber : EventSubscriberBase<HttpNotificationCreatedEvent>
    {
        private readonly IRemoteSchedulerDbContext _dbContext;
```

```
        private readonly ICommandBus _commandBus;
        public
ScheduleHttpNotificationEventSubscriber(IRemoteSchedulerDbCo
ntext dbContext, ICommandBus commandBus)
        {
            _dbContext = dbContext;
            _commandBus = commandBus;
        }
        public override async Task
Handle(HttpNotificationCreatedEvent @event)
        {
            var httpNotification = await _dbContext.HttpNotifications
                .Where(notification => notification.Id == @event.Id)
                .Select(notification => new
                {
                    Interval = notification.Job.Interval,
                    CronSchedule = notification.Job.CronSchedule,
                    JobId = notification.Job.Id,
                    RepeatsCount = notification.Job.RepeatsCount,
                    StartAt = notification.Job.StartAt,
                    UserId = notification.Job.OwnerId,
                    NotificationId = notification.Id,
                    notification.Path,
                    notification.Body,
                    notification.Schema,
                    notification.Domain,
                    notification.Headers,
                    notification.ActionType,
                    notification.QueryParameters
                }).FirstOrDefaultAsync();
            if (httpNotification == null)
            {
                throw new
EntityNotFoundException<HttpNotificationCreatedEvent,
HttpNotification>(@event);
            }
            var sendingCommand = new
SendHttpNotificationCommand
            {
                Body = httpNotification.Body,
                Domain = httpNotification.Domain,
                Headers =
httpNotification.Headers?.ToDictionary(header => header.Key,
header => header.Value),
                Path = httpNotification.Path,
                Scheme = httpNotification.Schema,
                ActionType = httpNotification.ActionType,
                NotificationId = httpNotification.NotificationId,
                QueryParameters =

httpNotification.QueryParameters?.ToDictionary(parameter =>
parameter.Key,
                    parameter => parameter.Value)
            };
            var scheduleCommandExecutionCommand = new
ScheduleCommandExecutionCommand
            {
                Command = sendingCommand,
                Interval = httpNotification.Interval,
                CronSchedule = httpNotification.CronSchedule,
                JobId = httpNotification.JobId,
                RepeatsCount = httpNotification.RepeatsCount,
                StartAt = httpNotification.StartAt,
                UserId = httpNotification.UserId
            };
            await
_commandBus.Execute(scheduleCommandExecutionCommand);
        }
    }
}
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using RemoteScheduler.Core.Cqrs.Abstractions.Buses;
using RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Events;
namespace
RemoteScheduler.Services.Api.Implementation.Handlers.Events.N
otifications.Http
{
    public class UnscheduleHttpNotificationEventSubscriber :
EventSubscriberBase<HttpNotificationCanceledEvent>
    {
        private readonly ICommandBus _commandBus;
        private readonly IRemoteSchedulerDbContext _dbContext;
        public
UnscheduleHttpNotificationEventSubscriber(ICommandBus
commandBus, IRemoteSchedulerDbContext dbContext)
        {
            _commandBus = commandBus;
            _dbContext = dbContext;
        }
        public override async Task
Handle(HttpNotificationCanceledEvent @event)
        {
            var unscheduleCommandExecutionCommand = await
_dbContext.Jobs.Where(job => job.Id == @event.JobId)
                .Select(job => new
UnscheduleCommandExecutionCommand
                {
                    CommandType = job.CommandType,
                    JobId = job.Id,
                    UserId = job.OwnerId
                }).FirstOrDefaultAsync();
            if (unscheduleCommandExecutionCommand == null)
            {
                throw new
EntityNotFoundException<HttpNotificationCanceledEvent,
Job>(@event);
            }
            await
_commandBus.Execute(unscheduleCommandExecutionCommand)
;
        }
    }
}
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using RemoteScheduler.Core.Cqrs.Abstractions.Buses;
using RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Events;
namespace
RemoteScheduler.Services.Api.Implementation.Handlers.Events.N
otifications.Http
{
    public class UpdateScheduledHttpNotificationEventSubscriber :
EventSubscriberBase<HttpNotificationUpdatedEvent>
    {
        private readonly IRemoteSchedulerDbContext _dbContext;
        private readonly ICommandBus _commandBus;
        public
UpdateScheduledHttpNotificationEventSubscriber(IRemoteSchedul
erDbContext dbContext, ICommandBus commandBus)
        {
            _dbContext = dbContext;
            _commandBus = commandBus;
        }
        public override async Task
Handle(HttpNotificationUpdatedEvent @event)
        {
            var httpNotification = await _dbContext.HttpNotifications
                .Where(notification => notification.Id ==
@event.NotificationId).Select(notification => new
                {
                    notification.Body,
                    notification.Domain,
                    notification.Headers,
                    notification.QueryParameters,
                    notification.Schema,
                    notification.Job.OwnerId,
                    notification.JobId,
                    notification.Path,
                    notification.ActionType
```

```
            }).FirstOrDefaultAsync();
        if (httpNotification == null)
        {
            throw new
EntityNotFoundException<HttpNotificationUpdatedEvent,
HttpNotification>(@event);
        }
        var sendHttpNotificationCommand = new
SendHttpNotificationCommand
        {
            Body = httpNotification.Body,
            Domain = httpNotification.Domain,
            Headers =
httpNotification.Headers?.ToDictionary(header => header.Key,
header => header.Value),
            Path = httpNotification.Path,
            Scheme = httpNotification.Schema,
            ActionType = httpNotification.ActionType,
            NotificationId = @event.NotificationId,
            QueryParameters =
httpNotification.QueryParameters?.ToDictionary(parameter =>
parameter.Key, parameter => parameter.Value)
        };
        var updateScheduledCommandExecutionCommand = new
UpdateScheduledCommandExecutionCommand
        {
            Command = sendHttpNotificationCommand,
            JobId = httpNotification.JobId,
            UserId = httpNotification.OwnerId
        };
        await
_commandBus.Execute(updateScheduledCommandExecutionCom
mand);
      }
   }
}
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using RemoteScheduler.Core.Exceptions;
namespace
RemoteScheduler.Services.Api.Implementation.Handlers.Queries
{
   public class UserDetailsQueryHandler :
ContextQueryHandlerBase<UserDetailsQuery,
UserDetailsResponse>
   {
      public
UserDetailsQueryHandler(IRemoteSchedulerDbContext
dbContext) : base(dbContext)
      {
      }
      public override async Task<UserDetailsResponse>
Handle(UserDetailsQuery query)
      {
         var detailsResponse = await DbContext.Users.Where(user
=> user.Id == query.UserId).Select(user =>
            new UserDetailsResponse
            {
               Email = user.Email,
               ApiKey = user.ApiKey,
               FirstName = user.FirstName,
               LastName = user.LastName,
               AllowedServerIps = user.AllowedServerIps
            }).FirstOrDefaultAsync();
         if (detailsResponse == null)
         {
            throw new EntityNotFoundException<UserDetailsQuery,
User>(query);
         }
         return detailsResponse;
      }
   }
}
using System.Linq;
using System.Threading.Tasks;
namespace
RemoteScheduler.Services.Api.Implementation.Handlers.Queries.A
uthorization
{
   public class
      UserApiKeyInfoByIdQueryHandler :
ContextQueryHandlerBase<UserApiKeyInfoByIdQuery,
UserApiKeyInfoByIdResponse>
   {
      public
UserApiKeyInfoByIdQueryHandler(IRemoteSchedulerDbContext
dbContext) : base(dbContext)
      {
      }
      public override async Task<UserApiKeyInfoByIdResponse>
Handle(UserApiKeyInfoByIdQuery query)
      {
         var userApiKeyInfoByIdResponse = await
DbContext.Users.Where(user => user.ApiKey ==
query.ApiKey).Select(
            user => new UserApiKeyInfoByIdResponse
            {
               UserId = user.Id,
               AllowedIps = user.AllowedServerIps
            }).FirstOrDefaultAsync();
         if (userApiKeyInfoByIdResponse == null)
         {
            throw new UserByApiKeyNotFoundException();
         }
         return userApiKeyInfoByIdResponse;
      }
   }
}
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
namespace
RemoteScheduler.Services.Api.Implementation.Handlers.Queries.A
uthorization
{
   public class
      UserByCredentialsQueryHandler :
ContextQueryHandlerBase<UserByCredentialsQuery,
UserByCredentialsResponse>
   {
      public
UserByCredentialsQueryHandler(IRemoteSchedulerDbContext
dbContext) : base(dbContext)
      {
      }
      public override async Task<UserByCredentialsResponse>
Handle(UserByCredentialsQuery query)
      {
         var userByCredentialsResponse = await DbContext.Users
            .Where(user => user.Email == query.Email &&
user.PasswordHash == query.PasswordHash).Select(user =>
               new UserByCredentialsResponse
               {
                  UserId = user.Id
               }).FirstOrDefaultAsync();
         if (userByCredentialsResponse == null)
         {
            throw new UserByCredentialsNotFoundException();
         }
         return userByCredentialsResponse;
      }
   }
}
using System.Linq;
using System.Threading.Tasks;
namespace
RemoteScheduler.Services.Api.Implementation.Handlers.Queries.A
uthorization
{
   public class
```

```
    UserByRefreshTokenQueryHandler :
ContextQueryHandlerBase<UserByRefreshTokenQuery,
UserByRefreshTokenResponse>
  {
    public
UserByRefreshTokenQueryHandler(IRemoteSchedulerDbContext
dbContext) : base(dbContext)
    {
    }
    public override async Task<UserByRefreshTokenResponse>
Handle(UserByRefreshTokenQuery query)
    {
      var userByRefreshTokenResponse = await
DbContext.UserRefreshTokens
        .Where(token => token.DeviceId == query.DeviceId &&
token.RefreshToken == query.RefreshToken).Select(
          token => new UserByRefreshTokenResponse
          {
            UserId = token.UserId
          }).FirstOrDefaultAsync();
      if (userByRefreshTokenResponse == null)
      {
        throw new UserByRefreshTokenNotFoundException();
      }
      return userByRefreshTokenResponse;
    }
  }
}
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
namespace
RemoteScheduler.Services.Api.Implementation.Handlers.Queries.A
uthorization
{
  public class UserRefreshTokenByIdQueryHandler :
ContextQueryHandlerBase<UserRefreshTokenByIdQuery,
    UserRefreshTokenByIdResponse>
  {
    public
UserRefreshTokenByIdQueryHandler(IRemoteSchedulerDbContex
t dbContext) : base(dbContext)
    {
    }
    public override async
Task<UserRefreshTokenByIdResponse>
Handle(UserRefreshTokenByIdQuery query)
    {
      var userRefreshTokenByIdResponse = await
DbContext.UserRefreshTokens.Where(token =>
          token.UserId == query.UserId && token.DeviceId ==
query.DeviceId)
        .Select(token => new UserRefreshTokenByIdResponse
        {
          RefreshToken = token.RefreshToken
        }).FirstOrDefaultAsync();
      if (userRefreshTokenByIdResponse == null)
      {
        throw new RefreshTokenNotFoundException();
      }
      return userRefreshTokenByIdResponse;
    }
  }
}
using System.Linq;
using System.Threading.Tasks;
namespace
RemoteScheduler.Services.Api.Implementation.Handlers.Queries.N
otifications.Http
{
  public class
    HttpNotificationsQueryHandler :
ContextQueryHandlerBase<HttpNotificationsQuery,
HttpNotificationsResponse>
  {
    public
HttpNotificationsQueryHandler(IRemoteSchedulerDbContext
dbContext) : base(dbContext)
    {
    }
    public override async Task<HttpNotificationsResponse>
Handle(HttpNotificationsQuery query)
    {
      var httpNotifications = await DbContext.HttpNotifications
        .Where(notification => notification.Job.OwnerId ==
query.UserId)
        .Select(notification => new
HttpNotificationsResponse.HttpNotification
        {
          Domain = notification.Domain,
          Headers = notification.Headers.ToDictionary(header
=> header.Key, header => header.Value),
          Id = notification.Id,
          Interval = notification.Job.Interval,
          Path = notification.Path,
          Scheme = notification.Schema,
          Status = notification.Job.Status,
          CronExpression = notification.Job.CronSchedule,
          IsCron = notification.Job.IsCron,
          JobId = notification.JobId,
          QueryParameters =

notification.QueryParameters.ToDictionary(parameter =>
parameter.Key,
              parameter => parameter.Value),
          RepeatsCount = notification.Job.RepeatsCount,
          StartAt = notification.Job.StartAt
        }).ToListAsync();
      return new HttpNotificationsResponse
      {
        Notifications = httpNotifications
      };
    }
  }
}
using System;
using System.Text;zation;
namespace
RemoteScheduler.Services.Api.Implementation.Services
{
  public class AccessTokenService : IAccessTokenService
  {
    private readonly IAccessTokenConfiguration
_tokenConfiguration;
    public AccessTokenService(IAccessTokenConfiguration
tokenConfiguration)
    {
      _tokenConfiguration = tokenConfiguration;
    }
    public string CreateToken<TPayload>(TPayload payload,
string customSecret) where TPayload : IAccessTokenPayload
    {
      payload.Lifetime = _tokenConfiguration.Lifetime.Ticks;
      payload.ExpirationDate = DateTime.UtcNow +
_tokenConfiguration.Lifetime;
      var secret = _tokenConfiguration.SecretKey +
customSecret;
      var secretBytes = Encoding.UTF8.GetBytes(secret);
      var token = JWT.Encode(payload, secretBytes,
JwsAlgorithm.HS512);
      return token;
    }
    public TPayload GetPayload<TPayload>(string token) where
TPayload : IAccessTokenPayload
    {
      try
      {
        return JWT.Payload<TPayload>(token);
      }
      catch (JoseException e)
      {
        throw new InvalidAccessTokenException(e);
```

```
            }
            catch (Exception e)
            {
                throw new AccessTokenDecodingException(e);
            }
        }
        public TPayload ValidateToken<TPayload>(string token, string customSecret) where TPayload : IAccessTokenPayload
        {
            var secret = _tokenConfiguration.SecretKey + customSecret;
            var secretBytes = Encoding.UTF8.GetBytes(secret);
            TPayload accessTokenPayload;
            try
            {
                accessTokenPayload = JWT.Decode<TPayload>(token, secretBytes);
            }
            catch (JoseException e)
            {
                throw new InvalidAccessTokenException(e);
            }
            catch (Exception e)
            {
                throw new AccessTokenDecodingException(e);
            }
            if (accessTokenPayload.Lifetime != _tokenConfiguration.Lifetime.Ticks
                || accessTokenPayload.ExpirationDate < DateTime.UtcNow)
            {
                throw new AccessTokenExpiredException(token);
            }
            return accessTokenPayload;
        }
    }
}
using System;
using System.Security.Cryptography;
using RemoteScheduler.Services.Api.Abstractions.Services.Hashing;
namespace RemoteScheduler.Services.Api.Implementation.Services.Hashing
{
    public class RandomStringGenerationService : IRandomStringGenerationService
    {
        public string Generate(int bytesLength)
        {
            using (var rngCryptoServiceProvider = new RNGCryptoServiceProvider())
            {
                var bytes = new byte[bytesLength];
                rngCryptoServiceProvider.GetBytes(bytes);
                var fromBase64String = Convert.ToBase64String(bytes);
                return fromBase64String;
            }
        }
    }
}
using System;
using System.Security.Cryptography;
using System.Text;
using RemoteScheduler.Services.Api.Abstractions.Services.Hashing;
namespace RemoteScheduler.Services.Api.Implementation.Services.Hashing
{
    public class Sha512HashingService : ISha512HashingService
    {
        public string GenerateHash(string source)
        {
            var sourceBytes = Encoding.UTF8.GetBytes(source);
            byte[] hashBytes;
            using (var sha512 = new SHA512CryptoServiceProvider())
            {
                hashBytes = sha512.ComputeHash(sourceBytes);
            }
            return Convert.ToBase64String(hashBytes);
        }
    }
}
using System;
using System.Reflection;
using Autofac;
using RabbitMQ.Client;
using RemoteScheduler.Core.Configuration;
namespace RemoteScheduler.Services.Notifications.Email
{
    public class EmailService : IDisposable
    {
        private static readonly IEnvironment Environment = new Environment();
        private IContainer _container;
        public void Start()
        {
            var builder = new ContainerBuilder();
            builder.RegisterModule<CqrsModule>();
            builder.RegisterModule<CqrsRmqModule>();
            builder.Register(context =>
                {
                    var rabbitMqConfiguration = context.Resolve<IRabbitMqConfiguration>();
                    return new ConnectionFactory
                    {
                        DispatchConsumersAsync = true,
                        HostName = rabbitMqConfiguration.Host,
                        UserName = rabbitMqConfiguration.UserName,
                        Password = rabbitMqConfiguration.Password,
                        VirtualHost = rabbitMqConfiguration.VirtualHost,
                        AutomaticRecoveryEnabled = rabbitMqConfiguration.AutoRecovery,
                        Port = 5672
                    };
                })
                .AsImplementedInterfaces()
                .SingleInstance();

            builder.RegisterModule<EmailServiceImplementationModule>();
            builder.RegisterModule<EnvironmentModule>();
            builder.RegisterModule<Log4NetConfigurationModule>();
            builder.RegisterModule<Log4NetLoggerModule>();

            builder.RegisterConfiguration(Environment.EnvironmentName,
                Assembly.GetExecutingAssembly(),
                typeof(ILog4NetConfiguration).Assembly,
                typeof(IRabbitMqRpcConfiguration).Assembly);
            _container = builder.Build();
        }
        public void Dispose()
        {
            _container.Dispose();
        }
    }
}
using System;
using System.Runtime.Loader;
using System.Threading.Tasks;
using log4net;
using Newtonsoft.Json.Linq;
namespace RemoteScheduler.Services.Notifications.Email
{
    class Program
    {
        private static readonly ILog Logger = LogManager.GetLogger(typeof(Program));
        private static async Task Main(string[] args)
        {
            var taskCompletionSource = new TaskCompletionSource<bool>();
            Console.CancelKeyPress += (sender, eventArgs) =>
            {
                taskCompletionSource.SetResult(result: true);
            };
```

```
        var service = new EmailService();
        AppDomain.CurrentDomain.ProcessExit += (sender,
eventArgs) => OnStop(service);
        AssemblyLoadContext.Default.Unloading += context =>
OnStop(service);
        try
        {
            service.Start();
            Logger.Info($"Service
{typeof(Program).Assembly.GetName().Name} started");
            await taskCompletionSource.Task.ContinueWith(task =>
OnStop(service));
        }
        catch (Exception e)
        {
            Logger.Fatal(new JObject
            {
                ["Message"] = $"Service
{typeof(Program).Assembly.GetName().Name} startup failed",
                ["ExceptionMessage"] = e.Message
            });
            throw;
        }
    }
    private static void OnStop(EmailService service)
    {
        Logger.Info($"Service
{typeof(Program).Assembly.GetName().Name} is shutting down");
        service.Dispose();
    }
  }
}
using Microsoft.Extensions.Configuration;
using
RemoteScheduler.Services.Notifications.Email.Abstractions.Config
urations;
using IConfiguration =
RemoteScheduler.Core.Configuration.IConfiguration;
namespace
RemoteScheduler.Services.Notifications.Email.Configurations
{
  public class RabbitMqConfiguration : IRabbitMqConfiguration,
IConfiguration
  {
    public IConfigurationSection ConfigurationSection { get; set;
}
    public string Host { get; set; }
    public string UserName { get; set; }
    public string Password { get; set; }
    public string VirtualHost { get; set; }
    public bool AutoRecovery { get; set; }
  }
}
using Microsoft.Extensions.Configuration;
using
RemoteScheduler.Services.Notifications.Email.Abstractions.Config
urations;
using IConfiguration =
RemoteScheduler.Core.Configuration.IConfiguration;
namespace
RemoteScheduler.Services.Notifications.Email.Configurations
{
  public class SmtpClientConfiguration :
ISmtpClientConfiguration, IConfiguration
  {
    public string SenderEmail { get; set; }
    public string Login { get; set; }
    public string Password { get; set; }
    public string Host { get; set; }
    public int Port { get; set; }
    public bool UseSsl { get; set; }
    public IConfigurationSection ConfigurationSection { get; set;
}
  }
}
namespace
RemoteScheduler.Services.Notifications.Email.Abstractions.Config
urations
{
  public interface IRabbitMqConfiguration
  {
    string Host { get; }
    string UserName { get; }
    string Password { get; }
    string VirtualHost { get; }
    bool AutoRecovery { get; }
  }
}
namespace
RemoteScheduler.Services.Notifications.Email.Abstractions.Config
urations
{
  public interface ISmtpClientConfiguration
  {
    string SenderEmail { get; set; }
    string Login { get; set; }
    string Password { get; set; }
    string Host { get; set; }
    int Port { get; set; }
    bool UseSsl { get; set; }
  }
}
using System;
using System.Collections.Generic;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Services.Notifications.Email.Contracts.Comman
ds
{
  public class SendEmailNotificationCommand : ICommand
  {
    public ICollection<Guid> ReceiversIds { get; set; }
    public Guid EmailTemplateId { get; set; }
    public Guid? EmailSettingsId { get; set; }
    public string Subject { get; set; }
  }
}
using Autofac;
using
RemoteScheduler.Core.Cqrs.Abstractions.Handlers.Commands;
namespace
RemoteScheduler.Services.Notifications.Email.Implementation
{
  public class EmailServiceImplementationModule : Module
  {
    protected override void Load(ContainerBuilder builder)
    {
      builder.RegisterAssemblyTypes(ThisAssembly)
        .AsClosedTypesOf(typeof(ICommandHandler<>))
        .AsImplementedInterfaces();
      builder.RegisterAssemblyTypes(ThisAssembly)
        .AsClosedTypesOf(typeof(IEventSubscriber<>))
        .AsImplementedInterfaces();
      builder.RegisterAssemblyTypes(ThisAssembly)
        .AsClosedTypesOf(typeof(IQueryHandler<,>))
        .AsImplementedInterfaces();
    }
  }
}
using System.Net;
using System.Net.Mail;
using System.Threading.Tasks;
namespace
RemoteScheduler.Services.Notifications.Email.Implementation.Ha
ndlers.Commands
{
  public class SendEmailNotificationCommandHandler :
CommandHandlerBase<SendEmailNotificationCommand>
  {
    private readonly ISmtpClientConfiguration
_smtpClientConfiguration;
```

```
        public
SendEmailNotificationCommandHandler(ISmtpClientConfiguratio
n smtpClientConfiguration)
        {
            _smtpClientConfiguration = smtpClientConfiguration;
        }
        public override async Task
Handle(SendEmailNotificationCommand command)
        {
            var client = new SmtpClient
            {
                Port = _smtpClientConfiguration.Port,
                Host = _smtpClientConfiguration.Host,
                Credentials = new
NetworkCredential(_smtpClientConfiguration.Login,
_smtpClientConfiguration.Password),
                EnableSsl = _smtpClientConfiguration.UseSsl
            };
            var mailMessage = new MailMessage(from:
_smtpClientConfiguration.SenderEmail,
to:"siydekukke@desoz.com")
            {
                Body = "test diploma message",
                Subject = "test diploma message",
                IsBodyHtml = true
            };
            await client.SendMailAsync(mailMessage);
            client.Dispose();
        }
    }
}
using System;
using System.Reflection;
using
RemoteScheduler.Services.Notifications.Http.Implementation;
using Environment =
RemoteScheduler.Core.Configuration.Environment;
namespace RemoteScheduler.Services.Notifications.Http
{
    public class HttpService : IDisposable
    {
        private static readonly IEnvironment Environment = new
Environment();
        private IContainer _container;
        public void Start()
        {
            var builder = new ContainerBuilder();
            builder.RegisterModule<CqrsModule>();
            builder.RegisterModule<CqrsRmqModule>();
            builder.Register(context =>
                {
                    var rabbitMqConfiguration =
context.Resolve<IRabbitMqConfiguration>();
                    return new ConnectionFactory
                    {
                        DispatchConsumersAsync = true,
                        HostName = rabbitMqConfiguration.Host,
                        UserName = rabbitMqConfiguration.UserName,
                        Password = rabbitMqConfiguration.Password,
                        VirtualHost = rabbitMqConfiguration.VirtualHost,
                        AutomaticRecoveryEnabled =
rabbitMqConfiguration.AutoRecovery,
                        Port = 5672
                    };
                })
                .AsImplementedInterfaces()
                .SingleInstance();

builder.RegisterModule<HttpServiceImplementationModule>();
            builder.RegisterModule<EnvironmentModule>();
            builder.RegisterModule<Log4NetConfigurationModule>();
            builder.RegisterModule<Log4NetLoggerModule>();

builder.RegisterConfiguration(Environment.EnvironmentName,
                Assembly.GetExecutingAssembly(),
                typeof(ILog4NetConfiguration).Assembly,
                typeof(IRabbitMqRpcConfiguration).Assembly);
            _container = builder.Build();
        }
        public void Dispose()
        {
            _container.Dispose();
        }
    }
}
using System;
using System.Runtime.Loader;
using Newtonsoft.Linq;
namespace RemoteScheduler.Services.Notifications.Http
{
    class Program
    {
        private static readonly ILog Logger =
LogManager.GetLogger(typeof(Program));
        private static async Task Main(string[] args)
        {
            var taskCompletionSource = new
TaskCompletionSource<bool>();
            Console.CancelKeyPress += (sender, eventArgs) =>
            {
                taskCompletionSource.SetResult(result: true);
            };
            var service = new HttpService();
            AppDomain.CurrentDomain.ProcessExit += (sender,
eventArgs) => OnStop(service);
            AssemblyLoadContext.Default.Unloading += context =>
OnStop(service);
            try
            {
                service.Start();
                Logger.Info($"Service
{typeof(Program).Assembly.GetName().Name} started");
                await taskCompletionSource.Task.ContinueWith(task =>
OnStop(service));
            }
            catch (Exception e)
            {
                Logger.Fatal(new JObject
                {
                    ["Message"] = $"Service
{typeof(Program).Assembly.GetName().Name} startup failed",
                    ["ExceptionMessage"] = e.Message
                });
                throw;
            }
        }
        private static void OnStop(HttpService service)
        {
            Logger.Info($"Service
{typeof(Program).Assembly.GetName().Name} is shutting down");
            service.Dispose();
        }
    }
}
using System.Collections.Generic;
namespace
RemoteScheduler.Services.Notifications.Http.Configurations
{
    public class HttpClientsConfiguration :
IHttpClientsConfiguration, IConfiguration
    {
        public Dictionary<string, string> DefaultHeaders { get; set; }
        public IConfigurationSection ConfigurationSection { get; set;
}
    }
}
using Microsoft.Extensions.Configuration;
namespace
RemoteScheduler.Services.Notifications.Http.Configurations
{
    public class RabbitMqConfiguration : IRabbitMqConfiguration,
IConfiguration
    {
```

```
        public IConfigurationSection ConfigurationSection { get; set;
}
        public string Host { get; set; }
        public string UserName { get; set; }
        public string Password { get; set; }
        public string VirtualHost { get; set; }
        public bool AutoRecovery { get; set; }
    }
}
using System.Collections.Generic;
namespace
RemoteScheduler.Services.Notifications.Http.Abstractions.Configu
rations
{
    public interface IHttpClientsConfiguration
    {
        Dictionary<string, string> DefaultHeaders { get; set; }
    }
}
namespace
RemoteScheduler.Services.Notifications.Http.Abstractions.Configu
rations
{
    public interface IRabbitMqConfiguration
    {
        string Host { get; }
        string UserName { get; }
        string Password { get; }
        string VirtualHost { get; }
        bool AutoRecovery { get; }
    }
}
using System.Collections.Generic;
using System.Net.Http;
namespace
RemoteScheduler.Services.Notifications.Http.Abstractions.Factorie
s
{
    public interface IHttpClientFactory
    {
        HttpClient CreateClient(IDictionary<string, string> headers =
null);
    }
}
using System;
using System.Collections.Generic;
namespace
RemoteScheduler.Services.Notifications.Http.Contracts.Commands
{
    public class SendHttpNotificationCommand : ICommand
    {
        public Guid NotificationId { get; set; }
        public string Scheme { get; set; }
        public string Domain { get; set; }
        public string Path { get; set; }
        public Dictionary<string, string> Headers { get; set; }
        public Dictionary<string, string> QueryParameters { get; set; }
        public HttpActionType ActionType{ get; set; }
        public JObject Body { get; set; }
    }
}
using Autofac;
namespace
RemoteScheduler.Services.Notifications.Http.Implementation
{
    public class HttpServiceImplementationModule : Module
    {
        protected override void Load(ContainerBuilder builder)
        {
            builder.RegisterAssemblyTypes(ThisAssembly)
                .AsClosedTypesOf(typeof(ICommandHandler<>))
                .AsImplementedInterfaces();
            builder.RegisterAssemblyTypes(ThisAssembly)
                .AsClosedTypesOf(typeof(IEventSubscriber<>))
                .AsImplementedInterfaces();
            builder.RegisterAssemblyTypes(ThisAssembly)
                .AsClosedTypesOf(typeof(IQueryHandler<,>))
                .AsImplementedInterfaces();
            builder.RegisterType<HttpClientFactory>()
                .AsImplementedInterfaces()
                .SingleInstance();
        }
    }
}
using System.Collections.Generic;
using System.Net.Http;
using
RemoteScheduler.Services.Notifications.Http.Abstractions.Configu
rations;
{
    internal class HttpClientFactory : IHttpClientFactory
    {
        private readonly IHttpClientsConfiguration
_clientsConfiguration;
        public HttpClientFactory(IHttpClientsConfiguration
clientsConfiguration)
        {
            _clientsConfiguration = clientsConfiguration;
        }
        public HttpClient CreateClient(IDictionary<string, string>
headers = null)
        {
            var httpClient = new HttpClient();
            foreach (var clientsConfigurationDefaultHeader in
_clientsConfiguration.DefaultHeaders)
            {

httpClient.DefaultRequestHeaders.Add(clientsConfigurationDefault
Header.Key, clientsConfigurationDefaultHeader.Value);
            }
            if (headers != null)
            {
                foreach (var headerPair in headers)
                {

httpClient.DefaultRequestHeaders.Add(headerPair.Key,
headerPair.Value);
                }
            }
            return httpClient;
        }
    }
}
using System;
using System.Linq;
using System.Net.Http;
using System.Net.Http.Formatting;
namespace
RemoteScheduler.Services.Notifications.Http.Implementation.Hand
lers.Commands
{
    public class SendHttpNotificationCommandHandler :
CommandHandlerBase<SendHttpNotificationCommand>
    {
        private static readonly JsonMediaTypeFormatter
JsonMediaTypeFormatter = new JsonMediaTypeFormatter
        {
            SerializerSettings = new JsonSerializerSettings
            {
                Formatting = Formatting.Indented,
                ContractResolver = new
CamelCasePropertyNamesContractResolver()
            }
        };
        private readonly IHttpClientFactory _httpClientFactory;
        public
SendHttpNotificationCommandHandler(IHttpClientFactory
httpClientFactory)
        {
            _httpClientFactory = httpClientFactory;
        }
        public override async Task
Handle(SendHttpNotificationCommand command)
        {
```

```
        var urlString = string.Concat(command.Scheme, "://",
command.Domain);
        if (command.Path != null)
        {
            urlString = string.Concat(urlString, "/", command.Path);
        }
        if (command.QueryParameters != null &&
command.QueryParameters.Any())
        {
            urlString = command.QueryParameters.Aggregate(
                new StringBuilder(string.Concat(urlString, "?")),
                (builder, pair) =>
builder.Append(string.Concat(Uri.EscapeDataString(pair.Key),
"=",
                    Uri.EscapeDataString(pair.Value), "&")),
                builder => builder.Remove(builder.Length - 1,
1).ToString());
        }
        var uri = new Uri(urlString);
        using (var httpRequestMessage = new
HttpRequestMessage(new
HttpMethod(command.ActionType.ToString()), uri))
        {
            if (command.Body != null)
            {
                httpRequestMessage.Content = new
ObjectContent<JObject>(command.Body,
JsonMediaTypeFormatter);
            }
            using (var httpClient =
_httpClientFactory.CreateClient(command.Headers))
            {
                try
                {
                    var httpResponseMessage = await
httpClient.SendAsync(httpRequestMessage);
                }
                catch (Exception ex)
                {
                }
            }
        }
    }
}
using System;
using Newtonsoft.Json.Linq;
namespace RemoteScheduler.Services.Scheduling
{
    class Program
    {
        private static readonly ILog Logger =
LogManager.GetLogger(typeof(Program));
        private static async Task Main(string[] args)
        {
            var taskCompletionSource = new
TaskCompletionSource<bool>();
            Console.CancelKeyPress += (sender, eventArgs) =>
            {
                taskCompletionSource.SetResult(result: true);
            };
            var service = new SchedulingService();
            AppDomain.CurrentDomain.ProcessExit += (sender,
eventArgs) => OnStop(service);
            AssemblyLoadContext.Default.Unloading += context =>
OnStop(service);
            try
            {
                service.Start();
                Logger.Info($"Service
{typeof(Program).Assembly.GetName().Name} started");
                await taskCompletionSource.Task.ContinueWith(task =>
OnStop(service));
            }
            catch (Exception e)
            {
                Logger.Fatal(new JObject
                {
                    ["Message"] = $"Service
{typeof(Program).Assembly.GetName().Name} startup failed",
                    ["ExceptionMessage"] = e.Message
                });
                throw;
            }
        }
        private static void OnStop(SchedulingService service)
        {
            Logger.Info($"Service
{typeof(Program).Assembly.GetName().Name} is shutting down");
            service.Dispose();
        }
    }
}
using System;
using System.Collections.Generic;
using RemoteScheduler.Services.Scheduling.Implementation;
using Environment =
RemoteScheduler.Core.Configuration.Environment;
namespace RemoteScheduler.Services.Scheduling
{
    internal sealed class SchedulingService : IDisposable
    {
        private static readonly IEnvironment Environment = new
Environment();
        private IContainer _container;
        public void Start()
        {
            var builder = new ContainerBuilder();
            builder.RegisterModule<CqrsModule>();
            builder.RegisterModule<CqrsRmqModule>();
            builder.Register(context =>
            {
                var rabbitMqConfiguration =
context.Resolve<IRabbitMqConfiguration>();
                return new ConnectionFactory
                {
                    DispatchConsumersAsync = true,
                    HostName = rabbitMqConfiguration.Host,
                    UserName = rabbitMqConfiguration.UserName,
                    Password = rabbitMqConfiguration.Password,
                    VirtualHost = rabbitMqConfiguration.VirtualHost,
                    AutomaticRecoveryEnabled =
rabbitMqConfiguration.AutoRecovery,
                    Port = 5672
                };
            })
            .AsImplementedInterfaces()
            .SingleInstance();

builder.RegisterModule<SchedulerImplementationModule>();
            builder.RegisterModule<EnvironmentModule>();
            builder.RegisterModule<Log4NetConfigurationModule>();
            builder.RegisterModule<Log4NetLoggerModule>();

builder.RegisterConfiguration(Environment.EnvironmentName,
                Assembly.GetExecutingAssembly(),
                typeof(ILog4NetConfiguration).Assembly,
                typeof(IRabbitMqRpcConfiguration).Assembly);

builder.RegisterType<JsonObjectSerializer>().AsImplementedInter
faces();
            builder.RegisterModule(new QuartzAutofacFactoryModule
            {
                ConfigurationProvider = context =>
                {
                    var quartzConfiguration =
context.ResolveOptional<IQuartzConfiguration>();
                    if (quartzConfiguration == null)
                    {
                        return new NameValueCollection();
                    }
                    var configurationDictionary =
quartzConfiguration.ConfigMap;
```

```
            var nameValueCollection =
configurationDictionary.Aggregate(new NameValueCollection(),
                (collection, pair) =>
                {
                    collection.Add(pair.Key, pair.Value);
                    return collection;
                });
            return nameValueCollection;
        }
    });
    builder.RegisterType<QuartzSchedulerManager>()
        .SingleInstance()
        .AsSelf()
        .AutoActivate()
        .OnActivated(args =>
args.Instance.Start().GetAwaiter().GetResult());
    _container = builder.Build();
  }
  public void Dispose()
  {
    _container?.Dispose();
  }
 }
}
using System.Collections.Generic;
using Microsoft.Extensions.Configuration;
using
RemoteScheduler.Services.Scheduling.Abstractions.Configurations
;
namespace RemoteScheduler.Services.Scheduling.Configurations
{
 public class QuartzConfiguration : IQuartzConfiguration
 {
  public IConfigurationSection ConfigurationSection { get; set;
}
  public Dictionary<string, string> ConfigMap { get; set; }
  public string SchedulerConnectionString { get; set; }
  public string InitializerConnectionString { get; set; }
 }
}
using Microsoft.Extensions.Configuration;
using
RemoteScheduler.Services.Scheduling.Abstractions.Configurations
;
namespace RemoteScheduler.Services.Scheduling.Configurations
{
 public class RabbitMqConfiguration : IRabbitMqConfiguration
 {
  public IConfigurationSection ConfigurationSection { get; set;
}
  public string Host { get; set; }
  public string UserName { get; set; }
  public string Password { get; set; }
  public string VirtualHost { get; set; }
  public bool AutoRecovery { get; set; }
 }
}
using System;
using System.Threading.Tasks;
namespace
RemoteScheduler.Services.Scheduling.QuartzManagement
{
 internal sealed class QuartzSchedulerManager : IDisposable
 {
  private readonly IScheduler _scheduler;
  private readonly IQuartzConfiguration _configuration;
  private readonly IEnvironment _environment;
  public QuartzSchedulerManager(IScheduler scheduler,
IQuartzConfiguration configuration,
    IEnvironment environment)
  {
   _scheduler = scheduler;
   _configuration = configuration;
   _environment = environment;
  }
  public async Task Start()
  {
    var dbName = new
NpgsqlConnectionStringBuilder(_configuration.SchedulerConnecti
onString).Database;
    using (var initializerConnection = new
NpgsqlConnection(_configuration.InitializerConnectionString))
    {
      var databaseExists = await
initializerConnection.QueryFirstOrDefaultAsync<bool>(

ScriptProvider.SqlScriptByName("CheckDatabaseExistence"), new
        {
          DatabaseName = dbName
        });
      if (databaseExists)
      {
        await _scheduler.Start();
        return;
      }
      await
initializerConnection.ExecuteAsync(string.Format(ScriptProvider.S
qlScriptByName("CreateDatabase"),
        dbName));
    }
    using (var schedulerConnection = new
NpgsqlConnection(_configuration.SchedulerConnectionString))
    {
      await
schedulerConnection.ExecuteAsync(ScriptProvider.SqlScriptByNa
me("InitializeSchema"));
    }
    await _scheduler.Start();
  }
  public void Dispose()
  {
    _scheduler?.Shutdown().GetAwaiter().GetResult();
  }
 }
}
using System;
namespace
RemoteScheduler.Services.Scheduling.QuartzManagement
{
 public static class ScriptProvider
 {
  public static string SqlScriptByName(string scriptName)
  {
    var assembly = Assembly.GetExecutingAssembly();
    using (var stream =
assembly.GetManifestResourceStream($"RemoteScheduler.Service
s.Scheduling.QuartzManagement.{scriptName}.sql"))
    {
      if (stream == null)
      {
        throw new ArgumentNullException(scriptName);
      }
      using (var reader = new StreamReader(stream))
      {
        return reader.ReadToEnd();
      }
    }
  }
 }
}
namespace
RemoteScheduler.Services.Scheduling.Abstractions.Configurations
{
 public interface IQuartzConfiguration : IConfiguration
 {
  Dictionary<string, string> ConfigMap { get; }
  string SchedulerConnectionString { get; }
  string InitializerConnectionString { get; }
 }
}
using RemoteScheduler.Core.Configuration;
namespace
RemoteScheduler.Services.Scheduling.Abstractions.Configurations
{
```

```
    public interface IRabbitMqConfiguration : IConfiguration
    {
        string Host { get; }
        string UserName { get; }
        string Password { get; }
        string VirtualHost { get; }
        bool AutoRecovery { get; }
    }
}
using System;
namespace
RemoteScheduler.Services.Scheduling.Abstractions.Jobs
{
    public interface IJobData
    {
        Guid JobId { get; }
    }
}
using Quartz;
namespace
RemoteScheduler.Services.Scheduling.Abstractions.Jobs
{
    public interface IJobDataSerializer
    {
        JobDataMap Serialize<TData>(TData data);
        TData Deserialize<TData>(JobDataMap map);
    }
}
using System.Threading.Tasks;
using Quartz;
namespace
RemoteScheduler.Services.Scheduling.Abstractions.Jobs
{
    public interface IJob<in TJobData> : IJob where TJobData :
IJobData, new()
    {
        Task Execute(TJobData jobData);
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
using RemoteScheduler.Services.Scheduling.Abstractions.Jobs;
namespace
RemoteScheduler.Services.Scheduling.Abstractions.Schedules
{
    public class ExecuteScheduledCommandJobData : IJobData
    {
        public ICommand Command { get; set; }
        public Guid JobId { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Services.Scheduling.Contracts.Commands
{
    public class ScheduleCommandExecutionCommand : ICommand
    {
        public Guid JobId { get; set; }
        public Guid UserId { get; set; }
        public DateTime StartAt { get; set; }
        public string CronSchedule { get; set; }
        public int RepeatsCount { get; set; }
        public TimeSpan? Interval { get; set; }
        public ICommand Command{ get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Buses;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Services.Scheduling.Contracts.Commands
{
    public class ScheduleEventTriggeringCommand : ICommand
    {
        public Guid JobId { get; set; }
        public string Group { get; set; }
        public DateTimeOffset Offset { get; set; }
        public DateTime StartAt { get; set; }
        public int TriggersCount { get; set; }
        public IEvent Event { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Services.Scheduling.Contracts.Commands
{
    public class UnscheduleCommandExecutionCommand :
ICommand
    {
        public Guid JobId { get; set; }
        public Guid UserId { get; set; }
        public string CommandType { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Services.Scheduling.Contracts.Commands
{
    public class UpdateScheduledCommandExecutionCommand :
ICommand
    {
        public Guid JobId { get; set; }
        public Guid UserId { get; set; }
        public ICommand Command { get; set; }
    }
}
using Autofac;act;
namespace RemoteScheduler.Services.Scheduling.Implementation
{
    public class SchedulerImplementationModule : Module
    {
        protected override void Load(ContainerBuilder builder)
        {
            builder.RegisterAssemblyTypes(ThisAssembly)
                .AsClosedTypesOf(typeof(ICommandHandler<>))
                .AsImplementedInterfaces();
            builder.RegisterAssemblyTypes(ThisAssembly)
                .AsClosedTypesOf(typeof(IEventSubscriber<>))
                .AsImplementedInterfaces();
            builder.RegisterAssemblyTypes(ThisAssembly)
                .AsClosedTypesOf(typeof(IQueryHandler<,>))
                .AsImplementedInterfaces();

builder.RegisterType<JobDataSerializer>().AsImplementedInterfac
es();
            builder.RegisterAssemblyTypes(ThisAssembly)
                .AsClosedTypesOf(typeof(IJob<>))
                .AsImplementedInterfaces();
        }
    }
}
using System;
using RemoteScheduler.Shared.Contracts.Events.Jobs;
namespace
RemoteScheduler.Services.Scheduling.Implementation.Handlers.C
ommands
{
    public class ScheduleCommandExecutionCommandHandler :
CommandHandlerBase<ScheduleCommandExecutionCommand>
    {
        private readonly IScheduler _scheduler;
        private readonly IJobDataSerializer _jobDataSerializer;
        private readonly IEventBus _eventBus;
        public
ScheduleCommandExecutionCommandHandler(IScheduler
scheduler, IJobDataSerializer jobDataSerializer,
            IEventBus eventBus)
        {
            _scheduler = scheduler;
            _jobDataSerializer = jobDataSerializer;
            _eventBus = eventBus;
```

```
    }
    public override async Task
Handle(ScheduleCommandExecutionCommand command)
    {
      var jobDataMap = _jobDataSerializer.Serialize(new
ExecuteScheduledCommandJobData
      {
        Command = command.Command,
        JobId = command.JobId
      });
      var commandType = command.Command.GetType().Name;
      var jobDetails =
JobBuilder.Create<IJob<ExecuteScheduledCommandJobData>>()
        .WithIdentity(command.JobId.ToString(),
$"{command.UserId}:{commandType}")
        .StoreDurably()
        .SetJobData(jobDataMap)
        .Build();
      var triggerKey = new
TriggerKey(command.JobId.ToString(),
$"{command.UserId}:{commandType}");
      var triggerBuilder = BuildTrigger(command, triggerKey);
      var trigger = triggerBuilder.Build();
      await _scheduler.ScheduleJob(jobDetails, trigger);
      await _eventBus.Raise(new JobScheduledEvent
      {
        JobId = command.JobId
      });
    }
    private static TriggerBuilder
BuildTrigger(ScheduleCommandExecutionCommand command,
TriggerKey triggerKey)
    {
      var triggerBuilder = TriggerBuilder.Create()
        .WithIdentity(triggerKey)
        .StartAt(command.StartAt);
      if (command.CronSchedule != null)
      {
        triggerBuilder =
triggerBuilder.WithCronSchedule(command.CronSchedule,
          builder => builder.InTimeZone(TimeZoneInfo.Utc)
            .WithMisfireHandlingInstructionFireAndProceed());
      }
      else
      {
        var scheduleBuilder =
SimpleScheduleBuilder.Create().WithMisfireHandlingInstructionN
owWithExistingCount();
        if (command.Interval.HasValue)
        {
          scheduleBuilder =
scheduleBuilder.WithInterval(command.Interval.Value);
        }
        if (command.RepeatsCount < 0)
        {
          scheduleBuilder = scheduleBuilder.RepeatForever();
        }
        else
        {
          scheduleBuilder =
scheduleBuilder.WithRepeatCount(command.RepeatsCount);
        }
        triggerBuilder =
triggerBuilder.WithSchedule(scheduleBuilder);
      }
      return triggerBuilder;
    }
  }
}
using System.Threading.Tasks;
using Quartz;
namespace
RemoteScheduler.Services.Scheduling.Implementation.Handlers.C
ommands
{
  public class UnscheduleCommandExecutionCommandHandler :
CommandHandlerBase<UnscheduleCommandExecutionCommand
>
  {
    private readonly IEventBus _eventBus;
    private readonly IScheduler _scheduler;
    public
UnscheduleCommandExecutionCommandHandler(IEventBus
eventBus, IScheduler scheduler)
    {
      _eventBus = eventBus;
      _scheduler = scheduler;
    }
    public override async Task
Handle(UnscheduleCommandExecutionCommand command)
    {
      var commandType = command.CommandType;
      var triggerKey = new
TriggerKey(command.JobId.ToString(),
$"{command.UserId}:{commandType}");
      var jobExists =
        await _scheduler.CheckExists(triggerKey);
      if (!jobExists)
      {
        throw new
EntityNotFoundException<UnscheduleCommandExecutionComma
nd, IJob>(command);
      }
      await _scheduler.UnscheduleJob(triggerKey);
      await _eventBus.Raise(new JobCanceledEvent
      {
        JobId = command.JobId
      });
    }
  }
}
using System.Threading.Tasks;
using Quartz;
using RemoteScheduler.Services.Scheduling.Contracts.Commands;
namespace
RemoteScheduler.Services.Scheduling.Implementation.Handlers.C
ommands
{
  public class
UpdateScheduledCommandExecutionCommandHandler :
CommandHandlerBase<UpdateScheduledCommandExecutionCom
mand>
  {
    private readonly IScheduler _scheduler;
    private readonly IJobDataSerializer _jobDataSerializer;
    public
UpdateScheduledCommandExecutionCommandHandler(ISchedule
r scheduler, IJobDataSerializer jobDataSerializer)
    {
      _scheduler = scheduler;
      _jobDataSerializer = jobDataSerializer;
    }
    public override async Task
Handle(UpdateScheduledCommandExecutionCommand command)
    {
      var jobDataMap = _jobDataSerializer.Serialize(new
ExecuteScheduledCommandJobData
      {
        Command = command.Command,
        JobId = command.JobId
      });
      var commandType = command.Command.GetType().Name;
      var triggerKey = new
TriggerKey(command.JobId.ToString(),
$"{command.UserId}:{commandType}");
      var jobExists =
        await _scheduler.CheckExists(triggerKey);
      if (!jobExists)
      {
        throw new
EntityNotFoundException<UpdateScheduledCommandExecutionC
ommand, IJob>(command);
```

```
        }
        var jobDetail = await _scheduler.GetJobDetail(new
JobKey(command.JobId.ToString(),
$"{command.UserId}:{commandType}"));
        jobDetail.JobDataMap["_jobData"] =
jobDataMap["_jobData"];
        await _scheduler.AddJob(jobDetail, true);
      }
   }
}
using System.Threading.Tasks;
using
RemoteScheduler.Services.Scheduling.Implementation.Jobs.Abstra
ct;
namespace
RemoteScheduler.Services.Scheduling.Implementation.Jobs
{
   public class ExecuteScheduledCommandJob :
JobHandlerBase<ExecuteScheduledCommandJobData>
   {
      private readonly ICommandBus _commandBus;
      public ExecuteScheduledCommandJob(IJobDataSerializer
jobDataSerializer,

ILogger<JobHandlerBase<ExecuteScheduledCommandJobData>>
logger, IEventBus eventBus, ICommandBus commandBus) : base(
        jobDataSerializer, logger, eventBus)
      {
        _commandBus = commandBus;
      }
      public override async Task
Execute(ExecuteScheduledCommandJobData jobData)
      {
        await _commandBus.Execute(jobData.Command);
      }
   }
}
using Newtonsoft.Json;
using Quartz;
using RemoteScheduler.Services.Scheduling.Abstractions.Jobs;
namespace
RemoteScheduler.Services.Scheduling.Implementation.Jobs.Abstra
ct
{
   public class JobDataSerializer : IJobDataSerializer
   {
      private static readonly string JobBaseKey = "_jobData";
      private static readonly JsonSerializerSettings SerializerSettings
= new JsonSerializerSettings
      {
        TypeNameAssemblyFormatHandling =
TypeNameAssemblyFormatHandling.Full,
        TypeNameHandling = TypeNameHandling.All,
        DateParseHandling = DateParseHandling.None
      };
      public JobDataMap Serialize<TData>(TData data)
      {
        var jsonString = JsonConvert.SerializeObject(data,
SerializerSettings);
        return new JobDataMap
        {
          [JobBaseKey] = jsonString
        };
      }
      public TData Deserialize<TData>(JobDataMap map)
      {
        var jsonString = map[JobBaseKey].ToString();
        return JsonConvert.DeserializeObject<TData>(jsonString,
SerializerSettings);
      }
   }
}
using System;
using System.Threading.Tasks;
using RemoteScheduler.Shared.Contracts.Events.Jobs;
namespace
RemoteScheduler.Services.Scheduling.Implementation.Jobs.Abstra
ct
{
   public abstract class JobHandlerBase<TJobDetails> :
IJob<TJobDetails> where TJobDetails : IJobData, new()
   {
      private readonly IJobDataSerializer _jobDataSerializer;
      private readonly ILogger<JobHandlerBase<TJobDetails>>
_logger;
      private readonly IEventBus _eventBus;
      protected JobHandlerBase(IJobDataSerializer
jobDataSerializer, ILogger<JobHandlerBase<TJobDetails>>
logger,
        IEventBus eventBus)
      {
        _jobDataSerializer = jobDataSerializer;
        _logger = logger;
        _eventBus = eventBus;
      }
      public abstract Task Execute(TJobDetails jobData);
      public async Task Execute(IJobExecutionContext context)
      {
        var jobId = Guid.Parse(context.JobDetail.Key.Name);
        var executionId = Guid.NewGuid();
        await _eventBus.Raise(new JobExecutionStartedEvent
        {
          JobId = jobId,
          ExecutionId = executionId
        });
        try
        {
          var jobDetails =
_jobDataSerializer.Deserialize<TJobDetails>(context.MergedJobD
ataMap);
          await Execute(jobDetails);
          await _eventBus.Raise(new JobExecutionSucceedEvent
          {
            JobId = jobId,
            ExecutionId = executionId
          });
        }
        catch (Exception e)
        {
          _logger.Error(new JObject
          {
            ["Message"] = "Job execution failure",
            ["Job"] = JToken.FromObject(context.JobDetail)
          }, e);
          await _eventBus.Raise(new JobExecutionFailedEvent
          {
            JobId = jobId,
            ExecutionId = executionId
          });
        }
      }
   }
}
namespace RemoteScheduler.Shared.Contracts.Enums
{
   public enum HttpActionType
   {
      Post = 1,
      Put = 2,
      Get = 3,
      Delete = 4,
      Patch = 5
   }
}
namespace RemoteScheduler.Shared.Contracts.Enums.Jobs
{
   public enum JobExecutionStatus
   {
      Started = 1,
      Failed = 2,
      Succeed = 3
   }
```

```
}
namespace RemoteScheduler.Shared.Contracts.Enums.Jobs
{
    public enum JobStatus
    {
        Created = 0,
        Scheduled = 1,
        Done = 2,
        Expired = 3,
        Canceled = 4
    }
}
namespace RemoteScheduler.Shared.Contracts.Enums.Jobs
{
    public enum JobType
    {
        HttpNotification = 1,
        SmtpNotification = 2
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace RemoteScheduler.Shared.Contracts.Events.Jobs
{
    public interface IJobEvent : IEvent
    {
        Guid JobId { get; set; }
    }
}
using System;
namespace RemoteScheduler.Shared.Contracts.Events.Jobs
{
    public class JobCanceledEvent : IJobEvent
    {
        public Guid JobId { get; set; }
    }
}
using System;
namespace RemoteScheduler.Shared.Contracts.Events.Jobs
{
    public class JobExecutionFailedEvent : IJobEvent
    {
        public Guid JobId { get; set; }
        public Guid ExecutionId { get; set; }
    }
}
using System;
namespace RemoteScheduler.Shared.Contracts.Events.Jobs
{
    public class JobExecutionStartedEvent : IJobEvent
    {
        public Guid JobId { get; set; }
        public Guid ExecutionId { get; set; }
    }
}
using System;
namespace RemoteScheduler.Shared.Contracts.Events.Jobs
{
    public class JobExecutionSucceedEvent : IJobEvent
    {
        public Guid JobId { get; set; }
        public Guid ExecutionId { get; set; }
    }
}
using System;
namespace RemoteScheduler.Shared.Contracts.Events.Jobs
{
    public class JobExpiredEvent : IJobEvent
    {
        public Guid JobId { get; set; }
    }
}
using System;
namespace RemoteScheduler.Shared.Contracts.Events.Jobs
{
    public class JobScheduledEvent : IJobEvent
    {
        public Guid JobId { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Shared.Contracts.Events.Notifications
{
    public class HttpNotificationCanceledEvent : IEvent
    {
        public Guid NotificationId { get; set; }
        public Guid JobId { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Shared.Contracts.Events.Notifications
{
    public class HttpNotificationCreatedEvent : IEvent
    {
        public Guid Id { get; set; }
    }
}
using System;
using RemoteScheduler.Core.Cqrs.Abstractions.Contracts;
namespace
RemoteScheduler.Shared.Contracts.Events.Notifications
{
    public class HttpNotificationUpdatedEvent : IEvent
    {
        public Guid NotificationId { get; set; }
    }
}
```